21世纪高等学校计算机专业实用规划教材

Android 系统开发与实践

王友钊 黄静 戴燕云 编著



清华大学出版社

21世纪高等学校计算机专业实用规划教材

Android系统开发与实践

王友钊　黄　静　戴燕云　编著

清华大学出版社
北　京

内 容 简 介

本书是以讨论 Android 系统平台为基础，并结合实例讲解基于 Android 系统平台的应用开发实践过程为主要内容的基础教程和实践指导类教材。书中全方位讲解 Google 开放移动应用平台 Android 的各种特性，深入探讨了应用程序的基本组件、界面布局的基础，结合 Internet 实现通讯录的设计和发送短信实例详细介绍了 Android 系统平台开发的步骤和方法，从而实现对 Android 系统平台开发的深入了解。本书共分为三大部分：第一部分为第 1 章到第 3 章，主要介绍了 Android 系统的起源和相关的基础知识，为后面章节学习的基础；第二部分为第 4 章到第 6 章，主要介绍了 Android 系统开发的入门、Android 应用程序的结构与开发，以及 Android 在传感器网络方面的应用，为 Android 系统平台开发的基础；第三部分为第 7 章，通过实例的剖析和讲解指导读者实现对 Android 系统平台的开发应用。

本书内容丰富、分类合理清晰，既可作为大专院校电子信息类专业本科生和研究生的基础教材使用，还可作为有 Android 系统平台开发需求的初学者，以及有一定 Android 系统平台基础的开发人员的学习参考书。

图书在版编目（CIP）数据

Android 系统开发与实践/王友钊，黄静，戴燕云编著. —北京：清华大学出版社，2013.4
（21 世纪高等学校计算机专业实用规划教材）
ISBN 978-7-302-31578-0

Ⅰ. ①A… Ⅱ. ①王… ②黄… ③戴… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2013)第 030628 号

责任编辑：魏江江 王冰飞
封面设计：
责任校对：白 蕾
责任印制：

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经 销：全国新华书店
开 本：185mm×260mm **印 张**：20 **字 数**：487 千字
版 次：2013 年 4 月第 1 版 **印 次**：2013 年 4 月第 1 次印刷
印 数：1～ 000
定 价： .00 元

产品编号：050602-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程(简称'质量工程')"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

本系列教材立足于计算机专业课程领域，以专业基础课为主、专业课为辅，横向满足高校多层次教学的需要。在规划过程中体现了如下一些基本原则和特点。

(1) 反映计算机学科的最新发展，总结近年来计算机专业教学的最新成果。内容先进，充分吸收国外先进成果和理念。

(2) 反映教学需要，促进教学发展。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，融合先进的教学思想、方法和手段，体现科学性、先进性和系统性，强调对学生实践能力的培养，为学生知识、能力、素质协调发展创造条件。

(3) 实施精品战略，突出重点，保证质量。规划教材把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现教学质量和教学改革成果的教材。

(4) 主张一纲多本，合理配套。专业基础课和专业课教材配套，同一门课程有针对不同层次、面向不同应用的多本具有各自内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源配套。

(5) 依靠专家，择优选用。在制定教材规划时要依靠各课程专家在调查研究本课程教

材建设现状的基础上提出规划选题。在落实主编人选时，要引入竞争机制，通过申报、评审确定主题。书稿完成后要认真实行审稿程序，确保出书质量。

繁荣教材出版事业，提高教材质量的关键是教师。建立一支高水平教材编写梯队才能保证教材的编写质量和建设力度，希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21 世纪高等学校计算机专业实用规划教材

联系人：魏江江 weijj@tup.tsinghua.edu.cn

前　　言

本书主旨是让阅读本书的读者可以找到 Android 系统的构建和应用的脉络。本书内容的构成主要分成三个部分：第一部分主要介绍 Android 系统的起源和相关的基础知识，属于技术基础章节；第二部分主要揭示 Android 平台开发的相关知识，属于开发基础章节；第三部分剖解使用 Android 平台的开发实例，属于指导开发的章节。

本书的形成得益于王友钊、黄静和戴燕云老师组成的团队对便携式移动产品等科研项目的研发准备，并结合近两年教授“嵌入式技术”课时注重不断积累而形成的。三位老师撰写和整理了书中的主要部分，特别感谢张琦研究生的文字整理和完善工作，感谢陈烨以毕业论文的形式完成的开发实例。

本书的作者期许通过本书知识、方法和实际应用的讲解，开启初学者阅读和研究 Android 系统的思维；期许有部分 Android 系统基础的读者能够依据本书获得一次 Android 平台开发实践体验；还期许有一定 Android 系统开发经验的阅读者通过本书建立可以进一步沟通的平台。

编　者

2013 年 2 月

目　　录

第 1 章 Android 技术基础

基于嵌入式技术的 Android 是目前增长速度最快的智能 3G 手机操作系统，其底层系统以 Linux 内核为基础，用 C 语言开发；中间层包括函数库 Library 和虚拟机 Dalvik，用 C++语言开发；上层应用软件，包括通话程序、短信程序等，用 Java 语言开发。本章介绍 Android 的技术基础。

1.1 Android 的嵌入式技术基础

1.1.1 嵌入式系统定义

嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。它一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户的应用程序 4 个部分组成，用于实现对其他设备的控制、监视或管理等功能。

嵌入式系统一般指非 PC 系统，它包括硬件和软件两部分。硬件包括处理器/微处理器、存储器及外设器件和 I/O 端口、图形控制器等。软件部分包括操作系统软件(OS)(要求实时和多任务操作)和应用程序编程。应用程序控制着系统的运作和行为；而操作系统控制着应用程序编程与硬件的交互作用。有时设计人员把这两种软件组合在一起。

1.1.2 嵌入式系统的特点

嵌入式系统同通用型计算机系统相比具有以下特点：

(1) 嵌入式系统通常是面向特定应用的嵌入式 CPU，与通用型计算机系统的最大不同就是嵌入式 CPU 大多工作在为特定用户群设计的系统中，它通常都具有功耗低、体积小、集成度高等特点，能够把通用 CPU 中许多由板卡完成的任务集成在芯片内部，从而有利于嵌入式系统设计趋于小型化，移动能力大大增强，跟网络的耦合也越来越紧密。

(2) 嵌入式系统是将先进的计算机技术、半导体技术和电子技术与各个行业的具体应用相结合后的产物。这就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。

(3) 嵌入式系统的硬件和软件都必须高效率地设计，量体裁衣、去除冗余，力争在同样的硅片面积上实现更高的性能，这样才能在具体应用中对处理器的选择更具有竞争力。

(4) 嵌入式系统和具体应用有机地结合在一起，其升级换代也是和具体产品同步进行的，因此嵌入式系统产品一旦进入市场就具有较长的生命周期。

(5) 为了提高执行速度和系统可靠性,嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中,而不是存储于磁盘等载体中。

(6) 嵌入式系统本身不具备自主开发能力,即使设计完成以后用户通常也是不能对其中的程序功能进行修改的,必须有一套开发工具和环境才能进行开发。

1.1.3 嵌入式操作系统简介

嵌入式系统是将计算机直接嵌入系统中,是信息 IT 的最终产品。它根据应用的要求将操作系统和功能软件集成于计算机硬件系统中,实现软件与硬件的一体化。嵌入式操作系统是随着嵌入式系统的发展而出现的。

嵌入式操作系统负责嵌入式系统的全部软件、硬件资源的分配、调度与控制协调等活动,通过装卸模块进行功能配置,体现所在系统的特征。现在可供嵌入式应用的操作系统有许多,例如 Windows CE、Palm Operation System、Black berry 和 EPOC 等。Palm Operation System、Black berry 和 EPOC 一般只应用在手持设备上。微软推出的 Windows 的嵌入式版本——Windows CE 和风靡一时的嵌入式 Linux 则适合工业环境应用。嵌入式操作系统与桌面系统的不同主要体现在实时性、可裁剪性和可靠性 3 个方面。

1.1.4 嵌入式操作系统的应用与前景

1. 嵌入式操作系统的应用

1) 处理器技术

处理器技术与实现系统功能的计算引擎结构有关,很多不可编程的数字系统也可以视为处理器,这些处理器的差别在于其面向特定功能的专用化程度导致其设计指标与其他处理器不同。

(1) 通用处理器。这类处理器可用于不同类型的应用。它的一个重要特征就是存储程序,由于设计者不知道处理器将会运行何种运算,所以无法用数字电路建立程序;另一个特征就是通用的数据路径,为了处理各类不同的计算,数据路径是通用的,其数据路径一般有大量的寄存器以及一个或多个通用的算术逻辑单元。设计者只需要对处理器的存储器编程来执行所需的功能,即设计相关的软件。在嵌入式系统中使用通用处理器具有设计指标上的一些优势,例如提前上市时间和 NRE(Non-Recurring Engineering)成本较低等。因为设计者只需编写程序,而不需要做任何数字设计,灵活性高,功能的改变通过修改程序进行即可;与自行设计处理器相比,数量小时单位成本较低。当然,这种方式也有一些设计指标上的缺陷,例如,数量大时的单位成本相对较高,因为数量大时,自行设计的 NRE 成本分摊下来,会降低单位成本;对于某些应用,性能可能很差;由于包含了非必要的处理器硬件,系统的体积和功耗可能变大。

(2) 单用途处理器。单用途处理器是设计用于执行特定程序的数字电路,也指协处理器、加速器、外设等。例如 JPEG 编码/解码器执行单一程序,压缩或解压缩视频信息。嵌入式系统设计者可通过设计特定的数字电路来建立单用途的处理器,也可以采用预先设计好的商品化的单用途处理器。在嵌入式系统中使用单用途处理器,在指标上有一些优缺点,这些优缺点与通用处理器基本相反,性能可能更好,体积与功率可能较小,数量大时的单位成本可能较低,而设计时间与 NRE 成本可能较高,灵活性较差,数量小时的单位成本较高,对

某些应用性能不如通用处理器。

(3) 专用处理器。即专用指令集处理器(ASIP),是一个可编程处理器,针对某一特定类型的应用进行最优化。这类特定应用具有相同的特征,如嵌入式控制、数字信号处理等。在嵌入式系统中使用 ASIP 可以在保证良好的性能、功率和大小的情况下提供更大的灵活性,但这类处理器仍需要昂贵的 NRE 成本建立处理器本身和编译器。单片机和数字信号处理器是两类应用广泛的 ASIP,数字信号处理器是一种针对数字信号进行常见运算的微处理器,而单片机是一种针对嵌入式控制应用进行最佳化的微处理器,通常控制应用中的常见外设如串行通信外设、定时器、计数器、脉宽调制器及数/模转换器等都集成到了微处理器芯片上,从而使得产品的体积更小、成本更低。

2) IC 技术

(1) 全定制/VLSI。在全定制 IC(Integrated Circuit,集成电路)技术中,需要根据特定的嵌入式系统的数字实现来优化各层,设计人员从晶体管的版图尺寸、位置、连线开始设计以达到芯片面积利用率高、速度快、功耗低的最优化性能,利用掩膜在制造厂生产实际芯片。全定制的 IC 设计也常称为 VLSI(超大规模集成电路)设计,具有很高的 NRE 成本、很长的制造时间,适用于大量或对性能要求严格的应用。

(2) 半定制 ASIC。半定制 ASIC(Application Specific Integrated Circuit,专用集成电路)是一种约束型设计方法,包括门阵列设计法和标准单元设计法。它是在芯片上制作好一些具有通用性的单元元件和元件组的半成品硬件,设计者仅需要考虑电路的逻辑功能和各功能模块之间的合理连接即可。这种设计方法灵活方便、性价比高,缩短了设计周期,提高了成品率。

(3) 可编程 ASIC。可编程器件中所有各层都已经存在,设计完成后,在实验室里即可烧制出设计的芯片,不需要 IC 厂家参与,开发周期显著缩短。可编程 ASIC 具有较低的 NRE 成本,单位成本较高,功耗较大,速度较慢。

3) 设计/验证技术

嵌入式系统的设计技术主要包括硬件设计技术和软件设计技术两大类。其中,硬件设计技术主要包括芯片级设计技术和电路板级设计技术两个方面。芯片级设计技术的核心是编译/综合、库/IP、测试/验证。编译/综合技术使设计者用抽象的方式描述所需的功能,并自动分析和插入实现细节;库/IP 技术将预先设计好的低抽象级实现用于高级;测试/验证技术确保每级功能正确,减少各级之间反复设计的成本。

2. 嵌入式操作系统的应用领域与前景

1) 工业控制

基于嵌入式芯片的工业自动化设备已获得长足的发展,目前已经有大量的 8、16、32 位嵌入式微控制器正在应用中,网络化是提高生产效率和产品质量、减少人力资源的主要途径,例如工业过程控制、数字机床、电力系统、电网安全、电网设备监测、石油化工系统等。就传统的工业控制产品而言,低端型采用的往往是 8 位单片机。但是随着技术的发展,32 位、64 位的处理器逐渐成为工业控制设备的核心,在未来几年内必将获得迅速发展。

2) 交通管理

在车辆导航、流量控制、信息监测与汽车服务方面,嵌入式系统技术已经获得了广泛应用,内嵌 GPS 模块、GSM 模块的移动定位终端已经在各种运输行业获得了成功使用。目前

GPS 设备已经从尖端产品进入了普通百姓家庭，只需要几千元就可以随时随地找到用户想要找到的位置。

3）信息家电

这将成为嵌入式系统最大的应用领域，冰箱、空调等的网络化、智能化将引领人们的生活步入一个崭新的空间，即使不在家也可以通过电话线、网络进行远程控制。在这些设备中，嵌入式系统将大有用武之地。

4）家庭智能管理系统

水、电、煤气表的远程自动抄表，安全防火、防盗系统，其中嵌有的专用控制芯片将代替传统的人工检查，并实现更高，更准确和更安全的性能。目前在服务领域，如远程点菜器等已经体现了嵌入式系统的优势。

5）POS 网络及电子商务

公共交通无接触智能卡（Contactless Smart Card，CSC）发行系统、公共电话卡发行系统、自动售货机、各种智能 ATM 终端将全面走入人们的生活，到时手持一卡就可以行遍天下。

6）环境工程与自然

水文资料实时监测、防洪体系及水土质量监测、堤坝安全、地震监测网、实时气象信息网、水源和空气污染监测等，在很多环境恶劣，地况复杂的地区，嵌入式系统将实现无人监测。

7）机器人

嵌入式芯片的发展将使机器人在微型化、高智能方面的优势更加明显，同时会大幅度降低机器人的价格，使其在工业领域和服务领域获得更广泛的应用。

这些应用中，可以着重于在控制方面的应用。就远程家电控制而言，除了开发出支持 TCP/IP 的嵌入式系统之外，家电产品控制协议也需要制订和统一，这需要家电生产厂家来做。同样的道理，所有基于网络的远程控制器件都需要与嵌入式系统之间实现接口，然后再由嵌入式系统来控制并通过网络实现控制。所以，开发和探讨嵌入式系统有着十分重要的意义。

1.2 Android 开发的 Linux 基础

嵌入式 Linux 是将日益流行的 Linux 操作系统进行裁剪修改，使之能在嵌入式计算机系统上运行的一种操作系统。嵌入式 Linux 既继承了 Internet 上无限的开放源代码资源，又具有嵌入式操作系统的特性。嵌入式 Linux 的特点是版权费免费，购买费用、媒介成本、技术支持免费，且全世界的自由软件开发者提供支持网络特性的功能免费，而且性能优异，软件移植容易，代码开放，有许多应用软件支持，应用产品开发周期短，新产品上市迅速。因为有许多公开的代码可以参考和移植，且得到实时性能 RT_Linux Hardhat Linux 等嵌入式 Linux 系统的支持，实时性能稳定，安全性好。

嵌入式系统由于硬件的限制，通常只具有极少的硬件资源，如主频较低的 CPU、较小的内存、小容量的固态电子盘芯片 DoC(Disk on Chip)或 DoM (Disk on Module)替代磁盘等。在使用电池的系统中，它还要实现低功耗以延长电池的使用时间的功能。

Linux 作为嵌入式操作系统是完全可行的，因为 Linux 提供了完成嵌入功能的基本内核和所需要的所有用户界面，能处理嵌入式任务和用户界面。Linux 可看作连续的统一体，它从一个具有内存管理、任务切换和时间服务及其他分拆的微内核到完整的服务器，支持所有的文件系统和网络服务。Linux 作为嵌入式系统，是一个带有很多优势的新成员，它对许多 CPU 和硬件平台都是易移植、稳定、功能强大、易于开发的。

嵌入式 Linux 系统需要 3 个基本元素：系统引导工具（用于机器加电后的系统定位引导）、Linux 微内核（内存管理、程序管理）和初始化进程。但如果要它成为完整的操作系统并且继续保持小型化，还必须加上硬件驱动程序、硬件接口程序和应用程序组。

1.2.1 Linux 目录结构及文件

1. 什么是 Linux

Linux 是一套免费使用和自由传播的类 UNIX 操作系统，它速度快，运行稳定，对硬件的配置要求低，兼具了其他操作系统的优点，最关键是可以免费使用，所以 Linux 得到了迅猛发展。

说到 Linux，不得不提起 UNIX。UNIX 的庞大支持基础和发行系统，使得它（指 UNIX）成为世界范围内最有影响和最广泛使用的操作系统之一。起初 UNIX 是作为小型机和大型机上的多任务系统而开发的，尽管它有一些含糊不清的接口和缺少标准化等缺点，但是它仍然很快地发展成为广泛使用的操作系统。许多计算机爱好者感到 UNIX 正是他们想要的，但是由于商业版 UNIX 非常昂贵，而且源代码是有专利的，所以很难在计算机爱好者中广泛使用。于是出现这样一群人，他们是一支由编程高手、业余计算机玩家、黑客组成的奇怪队伍，完全独立地开发出一个在功能上毫不逊色于商业 UNIX 操作系统的全新免费 UNIX 操作系统——Linux。

Linux 作为 PC 上的一种 32 位类 UNIX 操作系统是在 1991 年下半年出现的。当时年仅 21 岁的芬兰大学生 Linus Torvalds 写这个操作系统的时候是为了做一个试验，写一个比当时流行的 MINIX 操作系统具有更多功能、更成熟的小型操作系统。虽然最初的 Linux 系统很小，功能也不多，但是随着 Internet 的发展，Linux 系统也被来自世界各地的数以千计的人（高手）不断扩充和完善，今天的 Linux 在很多方面已经领先了商业性的 UNIX 系统，它可以运行在包括 Intel 处理器、Motorola 的 M68k 处理器及 DEC 的 Alphas 等多种硬件平台，是真正的多用户、多任务的 32 位操作系统。像现代 UNIX 操作系统那样，Linux 也具有虚拟内存、共享库、命令装载、执行代码之间共享的复制—执行—写盘页操作、恰当的内存管理和 TCP/IP 网络等。

Linux 是一个遵循 POSIX（Portable Operating System Interface，可移植操作系统接口）标准的免费操作系统，具有 BSD 和 SYSV 的扩展特性（表明其在外表和性能上同常见的 UNIX 非常相像，但是所有系统核心代码已经全部被重新编写了）。它的版权所有者是芬兰籍的 Linus Torvalds 先生和其他开发人员，并且遵循 GPL（GNU General Public License，GNU 通用公共许可证）声明。

Linux 的许多其他应用程序是由自由软件基金会（Free Software Foundation，FSF）开发的。全世界许多热心的使用者为 Linux 开发或者移植了许多应用程序，包括 X-Window、Emacs、TCP/IP 网络（包括 SLIP/PPP/ISDN）等，现在 Linux（包括内核和大量的应用程序）

光是执行程序已经达到 200MB 的规模,完全安装以后的规模将更大(大约 500MB)。

使用 Linux 可以在相对低价的 Intel ×86 硬件平台上实现高档系统才具有的性能,许多用户在运行 Linux 的×86 机器上使用 benchmarks 进行测试,发现可以和 SUN 和 Digital 公司的中型工作站的性能媲美。事实上不光是许多爱好者和程序员在使用 Linux,许多商业用户比如 Internet 服务供应商(ISP)也使用 Linux 作为服务器代替昂贵的工作站,这些服务器的最高纪录是经过 600 天的运行没有碰到一次系统崩溃。

在 Linux 上可以运行大多数 UNIX 程序,例如 TeX、X-Window 系统、GNU C/C++ 编译器等,让用户在家中就可以享受 UNIX 的全部功能。如今有越来越多的商业公司采用 Linux 作为操作系统。例如,科学工作者使用 Linux 来进行分布式计算;ISP 使用 Linux 配置 Internet 服务器、电话拨号服务器来提供网络服务;CERN(欧洲核子中心)采用 Linux 做物理数据处理;美国 1998 年最卖座的影片《泰坦尼克号》中的计算机动画设计就是在 Linux 平台上进行的。如今越来越多的商业软件公司宣布支持 Linux,如 Corel 和 Borland 公司等。在国外的大学中很多教授用 Linux 来讲授操作系统原理和设计。当然,对于大多数用户来说最重要的一点是可以在自己家中的计算机上进行 UNIX 编程,享受阅读操作系统全部源代码的乐趣。

Linux 作为功能强大、性能出众、稳定可靠的操作系统,吸引着越来越多的使用者来使用,测试修改使用者编写的软件中的错误。在短短的几年时间里 Linux 以超常的速度发展,目前已经变成一个拥有广大用户群的真正优秀的、值得信赖的操作系统。根据不精确的统计,全世界使用 Linux 操作系统的用户已经有数百万之多,这一数字还在以惊人的速度增加着,而且绝大多数是在网络上使用的。在我国,随着 Internet 大潮的卷入,一批主要以高校学生和 ISP 技术人员组成的 Linux 爱好者队伍也已经蓬勃地成长起来,而且随着网络的不断普及,免费而性能优异的 Linux 操作系统必将发挥出越来越大的作用。

2. Linux 的组成

Linux 一般有 4 个主要部分:内核(Kernel)、外壳程序(Shell)、文件结构(File System)和实用工具。

1) Linux 内核

内核是系统的心脏,是运行程序和管理如磁盘和打印机等硬件设备的核心程序。

2) Linux Shell

Shell 是系统的用户界面,提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。

实际上 Shell 是一个命令解释器,解释由用户输入的命令并且把它们送到内核。不仅如此,Shell 有自己的编程语言用于对命令的编辑,它允许用户编写由 shell 命令组成的程序。Shell 编程语言具有普通编程语言的很多特点,例如它也有循环结构和分支控制结构等,用这种编程语言编写的 Shell 程序与其他应用程序具有同样的效果。

Linux 提供了像 Microsoft Windows 那样的可视命令输入界面——X-Window 的图形用户界面(GUI)。它提供了很多窗口管理器,其操作就像 Windows 中的一样,有窗口、图标和菜单,所有的管理都是通过鼠标控制。现在比较流行的窗口管理器是 KDE 和 GNOME。

每个 Linux 系统的用户可以拥有自己的用户界面或 Shell,用以满足专门的 Shell 需要。

同 Linux 本身一样,Shell 也有多种不同版本。目前主要有下列版本的 Shell:

Bourne Shell：是贝尔实验室开发的。

BASH：是 GNU 的 BourneAgainShell，是 GNU 操作系统上默认的 Shell。

Korn Shell：是对 Bourne Shell 的发展，在大部分内容上与 Bourne Shell 兼容。

C Shell：是 SUN 公司 Shell 的 BSD 版本。

3）Linux 文件结构

文件结构是文件存放在磁盘等存储设备上的组织方法，主要体现在对文件和目录的组织上。目录提供了管理文件的一个方便而有效的途径。用户可以从一个目录切换到另一个目录，可以设置目录和文件的权限以允许或拒绝其他人对其进行访问，也可以设置文件的共享程度。

Linux 目录采用多级树形结构，如图 1-1 所示，用户可以浏览整个系统，可以进入任何一个已授权进入的目录，访问其中的文件。

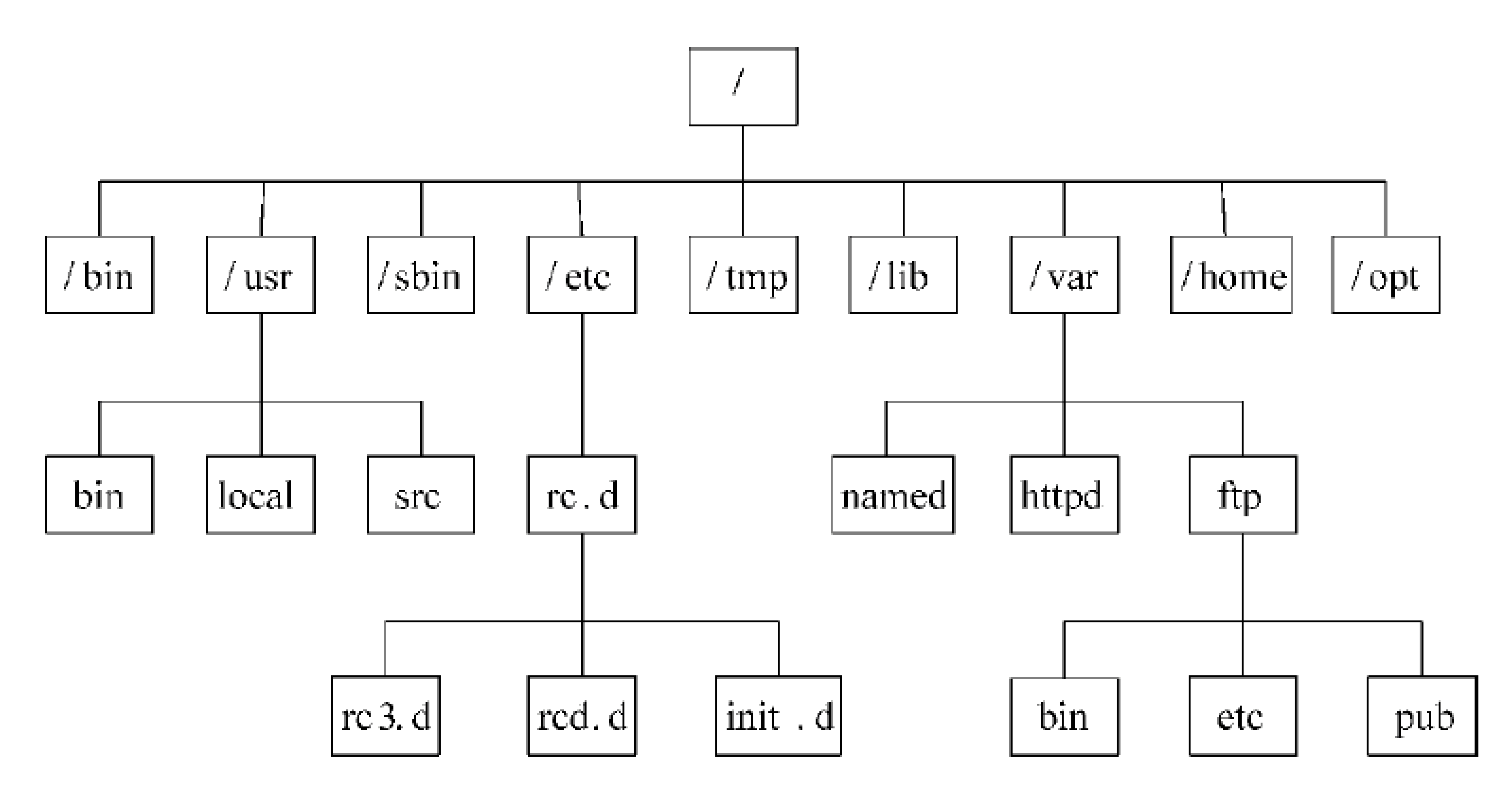


图 1-1 Linux 目录结构

文件结构的相互关联性使共享数据变得容易，几个用户可以访问同一个文件。Linux 是一个多用户系统，操作系统本身的驻留程序存放在以根目录开始的专用目录中，有时被指定为系统目录。图 1-1 中那些根目录下的目录就是系统目录。

4）Linux 实用工具

标准的 Linux 系统都有一套叫做实用工具的程序，它们是专门的程序，例如编辑器、执行标准的计算操作等。用户也可以产生自己的工具。Linux 的实用工具可分为以下三类。

(1) 编辑器：用于编辑文件。Linux 的编辑器主要有 Ed、Ex、Vi 和 Emacs。Ed 和 Ex 是行编辑器，Vi 和 Emacs 是全屏幕编辑器。

(2) 过滤器：用于接收数据并过滤数据。Linux 的过滤器(Filter)读取来自用户文件或其他地方的输入，检查和处理数据，然后输出结果。从这个意义上说，它们过滤了经过它们的数据。Linux 有不同类型的过滤器，一些过滤器用行编辑命令输出一个被编辑的文件；另外一些过滤器是按模式寻找文件并以这种模式输出部分数据；还有一些执行字处理操作，检测一个文件中的格式，输出一个格式化的文件。过滤器的输入可以是一个文件，也可以是用户从键盘输入的数据，还可以是另一个过滤器的输出。过滤器可以相互连接，因此一个过滤器的输出可能是另一个过滤器的输入。在有些情况下，用户可以编写自己的过滤器程序。

(3) 交互程序：允许用户发送信息或接收来自其他用户的信息。交互程序是用户与机器的信息接口。Linux 是一个多用户系统，它必须和所有用户保持联系。信息可以由系统上的不同用户发送或接收。信息的发送有两种方式，一种方式是与其他用户一对一地连接进行对话，另一种方式是一个用户对多个用户同时连接进行通信，即所谓广播式通信。

1.2.2 Linux 常用操作命令

1. su 命令

su 命令是最基本的命令之一，常用于不同用户间切换。例如，如果登录用户为 user1，要切换为 user2，则用如下命令：

```
$ su user2
```

然后系统提示输入 user2 密码，输入正确的密码之后就可以切换到 user2。完成之后就可以用 exit 命令返回到 user1。

su 命令的常见用法是切换至根用户或超级用户。如果发出不带用户名的 su 命令，则系统提示输入根用户密码，输入之后则可切换为根用户。

如果登录为根用户，则可以用 su 命令成为系统上任何用户而不需要密码。

2. pwd 命令

pwd 命令也是最常用、最基本的命令之一，用于显示用户当前所在的目录。

3. cd 命令

cd 命令不仅显示当前状态，还改变当前状态，它的用法跟 DOS 下的 cd 命令基本一致。

(1) cd..：可进入上一层目录。

(2) cd -：可进入上一次进入的目录。

(3) cd ～：可进入用户的 home 目录。

4. ls 命令

ls 命令跟 DOS 下的 dir 命令一样，用于显示当前目录的内容。

如果想取得详细信息，可用 ls-l 命令，这样就可以显示目录内容的详细信息。

如果目录下的文件太多，用一屏显示不了，可以用命令 ls -l |more 进行分屏显示。

5. find 命令

find 命令用于查找文件。这个命令可以按文件名、建立或修改日期、所有者(通常是建立文件的用户)、文件长度或文件类型进行搜索。

find 命令的基本结构如下：

```
$ find / - name X - print
```

其中，X 为文件名。

例如，要搜索系统上所有名称为 ye 的文件，可用如下命令：

```
$ find / - name ye - print
```

这样就可以显示出系统上所有名称为 ye 的文件。

6. tar 命令

tar 命令最初用于建立磁带备份系统，目前广泛用于建立文件发布档案。可用如下方法建立 tar 档案：

```
$ tar cvf
```

例如，如果要将当前目录中所有文件存档到 ye. tar 档案中，可用如下命令：

```
$ tar cvf ye.tar *.*
```

要浏览档案内容，则将 c 选项变成 t。例如要浏览 ye. tar 档案中的内容，可用如下命令：

```
$ tar tvf ye.tar
```

要取出档案内的内容，则将 c 选项变成 x。例如要将 ye. tar 档案中的内容取到当前目录中，可用如下命令：

```
$ tar xvf ye.tar
```

7. gzip 命令

gzip 命令用于压缩文件。例如，如果要将 ye. txt 文件压缩，可用如下命令：

```
$ gzip ye.txt
```

这样就可以压缩文件并在文件名后面加上 gz 扩展名，变成文件 ye. txt. gz。

解压缩文件可用 gzip -d 命令实现：

```
$ gzip -d ye.txt.gz
```

这样就可以解压缩文件并删除 gz 扩展名。除此之外还可以用 gunzip 命令来解压缩文件，效果跟用 gzip -d 命令一样。

旧版的 tar 命令不压缩档案，可用 gzip 压缩。例如：

```
$ tar cvf ye.tar *.txt
$ gzip ye.tar
```

则可建立压缩档案 ye. tar. gz。

新版的 tar 可以直接访问和建立 gzip 压缩的 tar 档案，只要在 tar 命令中加上 z 选项就可以了。例如：

```
$ tar czvf ye.tar *.txt
```

生成压缩档案 ye. tar. gz；

```
$ tar tzvf ye.tar *.txt
```

显示压缩档案 ye.tar.gz 的内容；

```
$ tar xzvf ye.tar *.txt
```

取出压缩档案 ye.tar.gz 的内容。

8. mkdir 命令

这个命令很简单，跟 DOS 的 md 命令用法几乎一样，用于建立目录。

9. cp 命令

cp 命令用于复制文件或目录。cp 命令可以一次复制多个文件，例如：

```
$ cp *.txt *.doc *.bak /home
```

将当前目录中扩展名为 txt、doc 和 bak 的文件全部复制到/home 目录中。

如果要复制整个目录及其所有子目录，可以用 cp -R 命令。

10. rm 命令

rm 命令用于删除文件或目录。

rm 命令会强制删除文件，如果想要在删除时提示确认，可用 rm -i 命令。

如果要删除目录，可用 rm -r 命令。rm -r 命令在删除目录时，每删除一个文件或目录都会显示提示。如果目录太大，响应每个提示是不现实的，这时可以用 rm -rf 命令来强制删除目录，这样即使用了-i 参数也当无效处理。

11. mv 命令

mv 命令用于移动文件和更名文件。例如：

```
$ mv ye.txt /home
```

将当前目录下的 ye.txt 文件移动到/home 目录下：

```
$ mv ye.txt ye1.txt
```

将 ye.txt 文件改名为 ye1.txt。

类似于 cp 命令的用法，mv 命令也可以一次移动多个文件，在此不再赘述。

12. reboot 命令

重启命令。

13. halt 命令

关机命令。

1.3 Android 开发的 Java 基础

在经历了以大型机为代表的集中计算模式和以 PC 为代表的分散计算模式之后，Internet 的出现使得计算模式进入了网络计算时代(异构时代)。网络计算模式的一个特点

是计算机是异构的，即计算机的类型和操作系统是不一样的。网络计算模式的另一个特点是代码可以通过网络在各种计算机上进行迁移。这就迫切需要一种跨平台的编程语言，使得用其编写的程序能够在网络中的各种计算机上正常运行，Java 就是在这种需求下产生的。

Java 是一种广泛使用的网络编程语言，是一种既面向对象又可跨平台的程序设计语言。Java 语言产生于 C++语言之后，充分吸取 C++语言的优点，采用程序员所熟悉的 C 和 C++语言的许多语法，同时又去掉了 C 语言中指针、内存申请和释放等影响程序健壮性的部分。可以说，Java 语言是站在 C++语言这个巨人的肩膀上前进的。

1.3.1 Java 语言的特点

SUN 公司的 Java 白皮书对 Java 做了如下定义：Java：Asimple，object-oriented，distributed，interpreted，robust，secure，architecture-neutral，portable，high-performance，multi-threaded，and dynamic language。

Java 是一种简单的、面向对象的、分布式的、解释执行的、健壮的、安全的、结构中立的、可移植的、高效率的、多线程的和动态的语言。Sun 公司对 Java 的定义充分展示了 Java 的以下几个特点。

1. 简单

Java 是一种简单的语言。Java 在 C 和 C++语言的基础上开发，继承了 C 和 C++语言的许多特性，同时也取消了 C 和 C++语言中烦琐的、难以理解的、不安全的内容，如指针、多重继承等。JDK 还提供了丰富的基础类库，具有 C 或 C++编程经验的程序员都会对这些基础类库很熟悉，无须经过长时间训练即可掌握它。

2. 面向对象

Java 是一种纯面向对象的语言。Java 面向对象的程序设计思路不同于 C 语言基于过程的程序设计思路。面向对象程序设计具备更好地模拟现实世界环境的能力和可重用性，它将待解决的现实问题转换成一组分离的程序对象，这些对象彼此之间可以进行交互，一个对象包含了对应实体应有的信息以及访问和改变这些信息的方法。通过这种设计方式所设计出来的程序更易于改进、扩展、维护和重用。Java 语言提供类、接口和继承等原语。Java 语言只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口间的实现机制。Java 语言全面支持动态绑定，而 C++语言只对虚函数使用动态绑定。

3. 分布式

Java 是一种分布式的语言。传统的基于 C/S(客户端/服务器)结构的程序，均采用客户端向服务器提出服务请求，服务器再根据要求执行适当的程序并将结果返回的方式，所以服务器负荷较重。Java 采用 Java 虚拟机结构，可将许多工作直接交由终端处理，因此数据可以被分布式处理。此外，Java 类库的运用大大减轻了网络传输的负荷。Java 类库包含了支持 HTTP 和 FTP 等基于 TCP/IP 协议的子库。Java 应用程序可凭借 URL 打开并访问网络上的对象，其访问方式与访问本地文件系统几乎完全相同。分布式网络环境提供了 Java 进一步发展的空间。

4. 高效解释执行

Java 是高效解释执行的语言。高级语言程序必须转换为机器语言程序才能在计算机

上执行。但是,不同的计算机系统所使用的机器语言不同。为了实现“一次编译,随处运行”的目标,Java 程序在编译时并不直接编译成特定的机器语言程序,而是编译成与系统无关的字节码(bytecode),由 Java 虚拟机(Java Virtual Machine,JVM)来执行。JVM 使得 Java 程序可以一次编译,随处运行,任何系统只要安装了 Java 虚拟机就可以执行 Java 程序。

JVM 能直接在任何机器上执行,为字节码提供运行环境。当 JVM 解释执行 Java 程序时,Java 实时编译器(Just-In-Time,JIT)会将字节码译成目标平台对应的机器语言的指令代码。

早先的许多尝试解决跨平台问题的方案对性能要求都很高。其他解释执行的语言系统如 BASIC、TCL、Perl 等都有无法克服的性能缺陷。但是,Java 却可以在非常低档的 CPU 上顺畅运行。这是因为 JVM 能够直接使用 JIT 编译技术将经过精心设计的字节码转换成高性能的本机代码。事实上,随着 JIT 编译器技术的发展,Java 程序的运行速度已接近于 C++语言。因而,“高效且跨平台”对 Java 来说已不再矛盾。

5. 健壮

Java 是健壮的语言。为了更好地理解 Java 的健壮性,先讨论一下传统编程环境下程序设计失败的主要原因:内存管理错误和误操作引起的异常或运行时异常。

在传统的编程环境下,内存管理是一项困难、乏味的工作。例如,在 C 或 C++语言中,必须手工分配、释放所有的动态内存。如果忘记释放原来分配的内存,或是释放了其他程序正在使用的内存时,系统就会出错。同时,在传统的编程环境下,异常情况可能经常由“被零除”、“Null 指针操作”、“文件未找到”等原因引起,必须用既烦琐又难理解的一大堆指令来进行处理。

Java 通过自行管理内存分配和释放的方法,从根本上消除了有关内存的问题。Java 提供垃圾收集器,可自动收集闲置对象占用的内存。Java 提供面向对象的异常处理机制来解决异常处理的问题。通过类型检查、Null 指针检测、数组边界检测等方法,在程序开发早期就发现程序的错误。

6. 安全

Java 是安全的网络编程语言。因为 Java 常被用于网络环境中,为此,Java 提供了一系列的安全机制以防恶意代码攻击,确保系统安全。Java 的安全机制分为多级,包括 Java 语言本身的安全性设计以及严格的编译检查、运行检查和网络接口级的安全检查。

Java 语言是强类型语言,每种类型都要求严格定义。首先,每个变量、表达式都有类型。其次,所有的数值传递,不管是直接的还是通过方法调用经由参数传递,都要进行类型相容性检查。有些语言没有自动进行数据类型相容性检查,或对冲突的类型进行转换的机制,Java 编译器对所有的表达式和参数都要进行类型相容性的检查,以确保类型是兼容的。任何类型的不匹配都是错误的,在编译完成之前,错误必须全部被纠正。此外,Java 摒弃指针类型和数据类型的隐式转换,对内存访问进行了严格的限制。Java 编译器在编辑期间并不分配内存,而是推迟到运行时由解释器决定,这样编程人员就无法通过指针来非法访问内存。在运行期间,Java 的运行环境提供了 4 级安全保障机制:字节码校验器、类装载器、运行时内存布局和文件访问限制。在网络接口级,用户可按自己的需要来设置网络访问权限。

此外,Java 的未来版本将采用公开密钥法以及其他加密技术,来核实从网络上传输过来的代码的源主机及该代码的完整性。

7. 结构中立

Java 是结构中立的语言。Java 的设计目标是支持网络应用。一般而言，网络是由许多不同的系统构成，包括各种不同的 CPU 与操作系统。为了让 Java 应用程序能够在网络上任何地方执行，其编译器会产生一种具备结构中立性的对象文件格式，即 Java 字节码文件。Java 字节码可在任何安装了 Java 虚拟机的平台上运行。

8. 可移植

Java 开发的程序具有可移植性。结构中立是确保程序可移植的必要条件，此外还需要很多其他条件的配合。Java 在可移植性方面做了许多工作。Java 语言规范中没有任何"同具体实现相关"的内容，这解决了所有可能会影响到 Java 可移植性方面的问题。例如，在 Windows 3.1 中整数(Integer)为 16 位，在 Windows 95 中整数为 32 位，在 DEC Alpha 中整数为 64 位，在 Intel 486 中整数为 32 位，即不同的操作系统和 CPU 对数据类型及长度都做了不同的定义，从而给程序的可移植性带来了一定的难度。Java 通过定义独立于平台的基本数据类型及其运算，使数据得以在任何硬件平台上保持一致。事实上，几乎所有的 CPU 都能支持以上数据类型，都支持 8～64 位整数格式的补码运算和单/双精度浮点运算。

9. 高效率

Java 是高效率的语言。每一次的版本更新，Java 在性能上均做出了改进。在历经数个版本变更后，Java 已经拥有与 C/C++ 语言同样甚至更好的运行性能。如果解释器速度不慢，Java 可以在运行时直接将目标代码翻译成机器指令。使用 JVM 一秒钟内可调用 300 000 个过程，与 C/C++语言不相上下。

10. 多线程

Java 是支持多线程的语言。多线程是一种应用程序设计方法。线程是从大进程里分出来的、小的、独立的进程，使得在一个程序里可同时执行多个小任务。多线程带来的好处是具有更好的交互性能和实时控制性能。但采用传统的程序设计语言(如 C/C++ 语言)实现多线程非常困难。Java 实现了多线程技术，提供了一些简便地实现多线程的方法，并拥有一套高复杂性的同步机制。

11. 动态

Java 语言具有动态特性。Java 动态特性是其面向对象设计方法的扩展，允许程序动态地装入运行过程中所需的类，这是 C++语言进行面向对象程序设计所无法实现的。C++程序设计过程中，每当在类中增加一个实例变量或一种成员函数后，引用该类的所有子类都必须重新编译，否则将导致程序出错。Java 采取如下措施来解决此类问题：

(1) Java 编译器不是将对实例变量和成员函数的引用编译为数值引用，而是将符号引用信息在字节码中保存后传递给解释器，再由解释器在完成动态连接类后，将符号引用信息转换为数据偏移量。存储器生成的对象不在编译过程中决定，而是延迟到运行时由解释器确定。这样，对类中变量和方法进行更新时就不至于影响现存的代码。解释执行字节码时，这种符号信息的查找和转换过程仅在一个新的名字出现时才进行一次，随后代码便可以全速执行。

(2) 在运行时确定引用的好处是可以使用已被更新的类，而不必担心会影响原有的代码。如果程序连接了网络中另一系统的某一类，该类的所有者也可以自由地对该类进行更新，而不会使任何引用该类的程序崩溃。

(3) Java 还简化了使用一个升级的或全新的程序的方法。如果系统运行 Java 程序时遇到了不知如何处理的程序,Java 能自动下载所需要的功能程序。

Java 是一种比 C/C++语言更具动态特性的语言,在设计上强调为运行中的运算环境提供动态支持。Java 在运行时为模块与模块之间建立连接,并能够更直接地运用面向对象设计体系;程序库可以自由地增加新方法和实例变量,而不会对它们的用户产生任何影响。

1.3.2 Java 应用分类

Java 是类似 C++语言的一种计算机编译语言。网页上用的 Java Applets 属于 Java。需要注意的是,Java 与 JavaScript 完全是两种不同的东西,千万不要混为一谈。

Java 应用非常广。Java 分为 J2SE、J2EE 和 J2ME。J2SE 用于应用程序开发,而 J2EE 用于企业级开发,J2ME 用于嵌入式开发。在编程方面,Java 和微软的.NET 都是编程界中最卓越的,但是 Java 优越于其他开发语言之处在于它是开源的,若想知道更深的东西,只要肯学、肯花时间就行,而.NET 虽然简单但许多东西都是封装起来的。下面介绍 Java 的三类应用。

1. J2SE

J2SE 是 Java SE(Java Platform,Standard Edition,Java 平台标准版)的简称。它允许开发和部署在桌面、服务器、嵌入式环境和实时环境中使用的 Java 应用程序。Java SE 包含了支持 Java Web 服务开发的类,并为 Java EE 提供基础。

2. J2EE

J2EE 是 Java EE(Java Platform,Enterprise Edition,Java 平台企业版)的简称。企业版本帮助开发和部署可移植、健壮、可伸缩且安全的服务器端 Java 应用程序。Java EE 是在 Java SE 的基础上构建的,它提供 Web 服务、组件模型、管理和通信 API,可以用来实现企业级的面向服务体系结构(Service-Oriented Architecture,SOA)和 Web 2.0 应用程序。

3. J2ME

J2ME 即 Java ME(Java Platform Micro Edition,Java 平台微型版),是一种高度优化的 Java 运行环境,针对市面上的大量消费类电子设备,例如 Papers、Cellularphones(蜂窝电话)、Screen-Phones(可视电话)、Digital Set-Top Boxes(数字机顶盒)、Car Navigation Systems(汽车导航系统)等。J2ME 技术是在 1999 年的 JavaOne Developer Conference 大会上推出的,它将 Java 语言中与平台无关的特性移植到小型电子设备上,允许移动无线设备之间共享应用程序。Java ME 为在移动设备和嵌入式设备(例如手机、PDA、电视机顶盒和打印机)上运行的应用程序提供一个健壮且灵活的环境。Java ME 包括灵活的用户界面、健壮的安全模型、许多内置的网络协议以及对可以动态下载的联网和离线应用程序的丰富支持。基于 Java ME 规范的应用程序只需编写一次就可以用于许多设备,而且可以利用每个设备的本机功能。

1.3.3 Java 技术三大特性

1. Java 虚拟机

Java 虚拟机(JVM)在 Java 编程里面具有非常重要的地位,相当于前面提到的 Java 运行环境。

1）JVM 的概念

JVM 是在真实机器中用软件模拟实现的一种想象机器。JVM 为不同的硬件平台提供了一种编译 Java 技术代码的规范，该规范使 Java 软件独立于平台，因为编译是针对作为虚拟机的“一般机器”而进行的。

2）JVM 的内容

JVM 为下列各项做出了定义：

（1）指令集（相当于 CPU）。

（2）寄存器。

（3）类文件夹。

（4）栈。

（5）垃圾收集堆。

（6）存储区。

3）JVM 的功能

（1）通过 ClassLoader 寻找和装载 class 文件。

（2）解释字节码成为指令并执行，提供 class 文件的运行环境。

（3）进行运行期间垃圾回收。

（4）提供与硬件交互的平台。

（5）JVM 是 Java 平台无关的保障。

正是因为有 JVM 这个中间层，Java 才能够实现与平台无关。JVM 就好比是一个 Java 运行的基本平台，所有的 Java 程序都运行在 JVM 上，所有与平台有关的东西都是由 JVM 去处理。

2. 垃圾回收

1）垃圾

在程序运行过程中，存在被分配了的内存块不再被需要的情况，那么这些内存块对程序来讲就是垃圾。

产生了垃圾，自然就需要清理这些垃圾，更为重要的是需要把这些垃圾所占用的内存资源回收回来加以再利用，从而节省资源、提高系统性能。

2）垃圾回收

对不再需要的已分配内存应取消分配，也就是释放内存，这个过程就是垃圾回收。

在其他语言中，取消分配是程序员的责任。Java 编程语言提供了一种系统级线程以跟踪内存分配，从而可以自动检查和释放不再需要的内存。

注意：在 Java 中，垃圾回收是一个自动的系统行为，程序员不能控制垃圾回收的功能和行为，例如垃圾回收什么时候开始，什么时候结束，到底哪些资源需要回收等，都不是程序员能控制的；有一些跟垃圾回收相关的方法，例如 System. gc()，调用这些方法仅仅是通知垃圾回收程序，至于垃圾回收程序运不运行、什么时候运行，都是程序员无法控制的；程序员可以通过设置对象为 null 来标识某个对象不再被需要了，这只是表示这个对象可以被回收了，而并不是马上被回收。

3）内存泄露

内存泄露就是程序运行期间所占用的内存一直往上涨，这很容易造成系统资源耗尽而

降低性能或崩溃。

3. 代码安全

Java 如何保证编写的代码是安全可靠的呢？

(1) 编写的代码首先要被编译成 class 文件，如果代码写得有问题，编译期间就会发现，然后提示有编译错误，就无法编译通过。

(2) 通过编译关后，在类装载的时候还会进行类装载检查，把本机上的类和网络资源类相分离，在调入类的时候进行检查，因而可以限制任何特洛伊木马的应用。

(3) 类装载后，在运行前还会进行字节码校验，以判断程序是安全的。

(4) 如果程序在网络上运行，还有沙箱(sandbox)的保护。沙箱就是指如果程序没有获得授权，只能在沙箱限定的范围内运行，是不能够访问本地资源的，从而保证了安全性。

1.3.4 Java 在 Android 平台开发中的应用

本节主要对 Java 程序进行简单介绍，以便能应用在 Android 平台的开发中。

1. Java 程序框架结构简介

Java 程序框架结构如图 1-2 所示，主要由以下几部分组成。

```
package com.axt.ch01;(声明包是 Java 程序框架的第一步，必须写在最开始。)
   public class HelloJava{ (本例中没有用到声明包，但以后我们编写的 Java
 程序都会用到声明包这行代码。)
         /**
          * 这是文档注释语句   (注释语句)
          */
         public static void main(String[] args){    (main 方法)
             System.out.println("Hello Java"); //在控制台显示字符串 Hello Java
(注释语句)
                     (main 下整块为代码输入区域)
         }
    }
```

图 1-2　Java 程序框架结构示意图

1) 声明包名

package 是声明包的关键字。com. axt. ch01 是包名，这里可以把 Java 的包理解为文件夹。在磁盘上也确实是 com\axt\ch01 的文件夹嵌套。

2) 外层框架

这里将其称为外层框架，以后会详细介绍这段代码的真正含义。public class 后面的 HelloJava 是类名，类名必须与文件的主文件名完全相同。本行最后有一个大括号，这个大括号与图中最后一行的大括号相匹配，缺一不可。

注释语句是给程序员看的，程序执行时，注释不会显示给用户看。注释语句的主要作用是帮助程序员理解、回忆代码的作用、实现的方法。

在实际开发中，注释语句起到了备忘录的重要作用，方便程序员进行代码的维护。

本例实现的注释有以下几个格式：

(1) /** */：多行文档注释，多行注释，允许在/** 和 */之间书写多行注释文字。文档注释中的文字可制作为 HTML 文档，方便程序员或用户查看。

(2) //：单行注释。注释的文字在一行内书写。

(3) /* */多行文档注释：允许在/* */之间书写多行注释文字。

3) 内层框架

main 是 Public static void main(String[]args){程序执行的起始处，这里暂时称其为程序的内层框架。以后会详细介绍这行程序中各部分的作用。本行最后的大括号与倒数第二行的大括号匹配，缺一不可。

4) 代码输入区域

程序员在代码输入区域编写 Java 程序。System.out.println 是一个 Java 命令，该命令向控制台显示一个字符串。字符串由双引号包括起来，并放在一对小括号中。Java 规定：每个命令结束时，由一个分号做标志。

2. Eclipse 开发工具

Eclipse 是目前最主流的 Java 开发工具。Java 开发分 3 个方向：JSE(Java Standard Enviroment，Java 标准开发环境)；JEE(Java Enterprise Environment，Java 企业开发环境)；JME(Java Mobile Environment，Java 移动开发环境)。这里介绍的 Eclipse 工具是 JEE。

1) Eclipse 简介

Eclipse 是著名的跨平台的自由集成开发环境(Integrated Development Environment，IDE)，最初主要用于 Java 语言开发，目前亦有人通过插件使其作为其他计算机语言(例如 C++语言)的开发工具。Eclipse 本身只是一个框架平台，但是众多插件的支持使得 Eclipse 拥有其他功能相对固定的 IDE 软件很难具有的灵活性。许多软件开发商以 Eclipse 为框架开发自己的 IDE。

Eclipse 最初是由 IBM 公司开发的，2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会(Eclipse Foundation)管理。

Eclipse 软件可以免费下载，以下重点介绍 Eclipse 的使用(Eclipse 3.6 版本)。

2) Eclipse 集成开发环境

Eclipse 集成开发环境界面如图 1-3 所示。

(1) 标题栏：显示 Eclipse 图标和当前正在编辑的 Java 程序的文件名，右上角还有关闭、最大化、最小化按钮。

(2) 主菜单：包含所有的 Eclipse 命令。以后会随用随讲，介绍常用的操作命令。

(3) 工具栏：包含最常用的操作命令，如运行、单步运行、调试等。

(4) 代码编辑区：这是编辑 Java 代码的区域。

(5) Package Explorer：包资源管理器，用来管理 Java 项目(开发的每个 Java 应用软件都被视为一个项目，每个项目中可包含众多的 Java 文件)。常用的还有 Navigator(导航器)。

(6) 控制台：显示文本结果的窗口，所有编写的基于显示文本的 Java 程序都在控制台

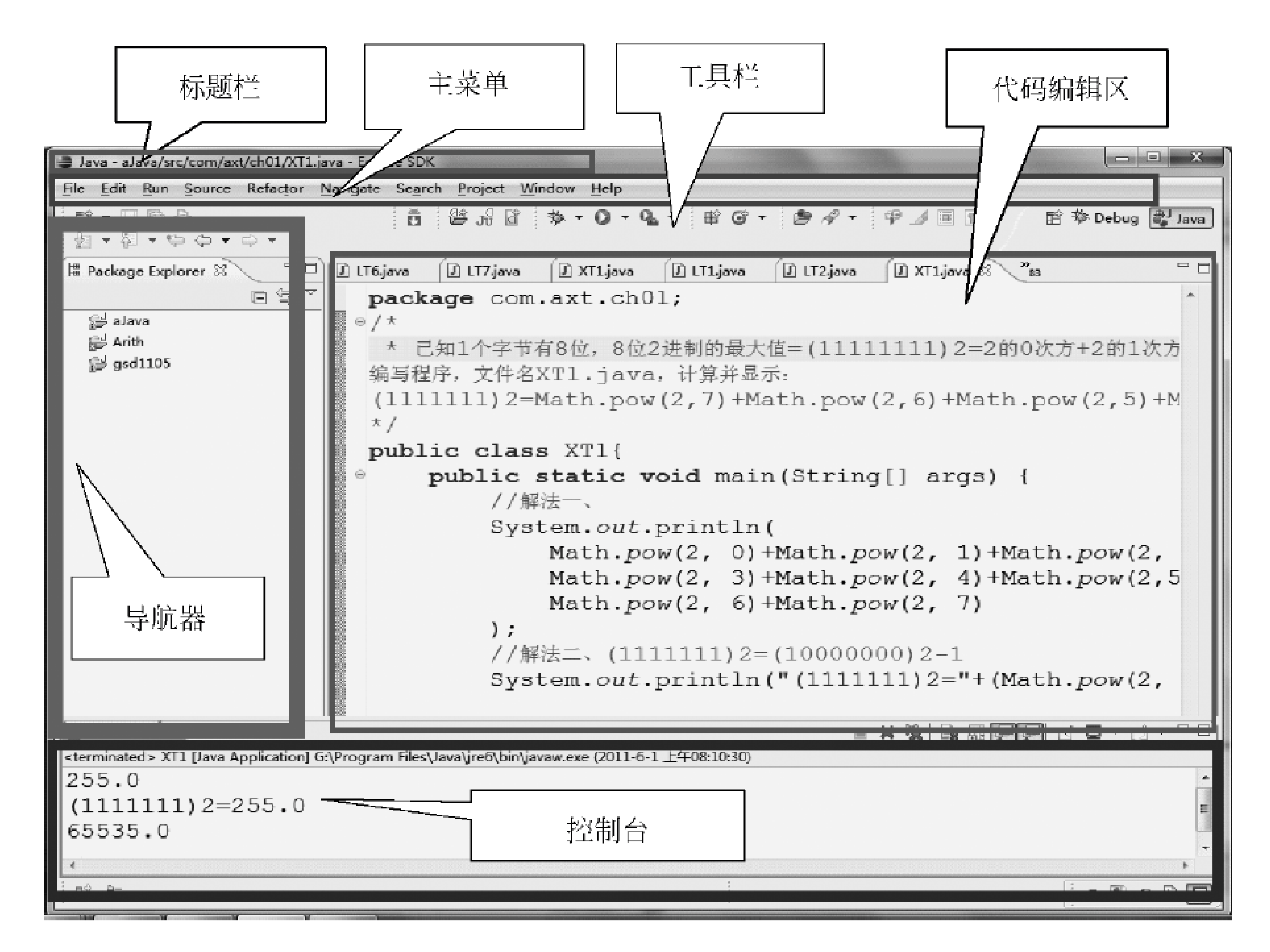


图 1-3　Eclipse 集成开发环境界面

输出程序的运行结果。

3）新建项目(Project)

Java 的每个应用程序都被视为一个项目(Project)，每个项目可包含众多程序。

例 1-1　按以下步骤创建一个名为 ajava 的项目。

(1) 在 Package Explorer(包资源管理器)中的空白处右击，如图 1-4 中的①所示；在弹出的快捷菜单中单击 New 命令，如图 1-4 中的②所示。

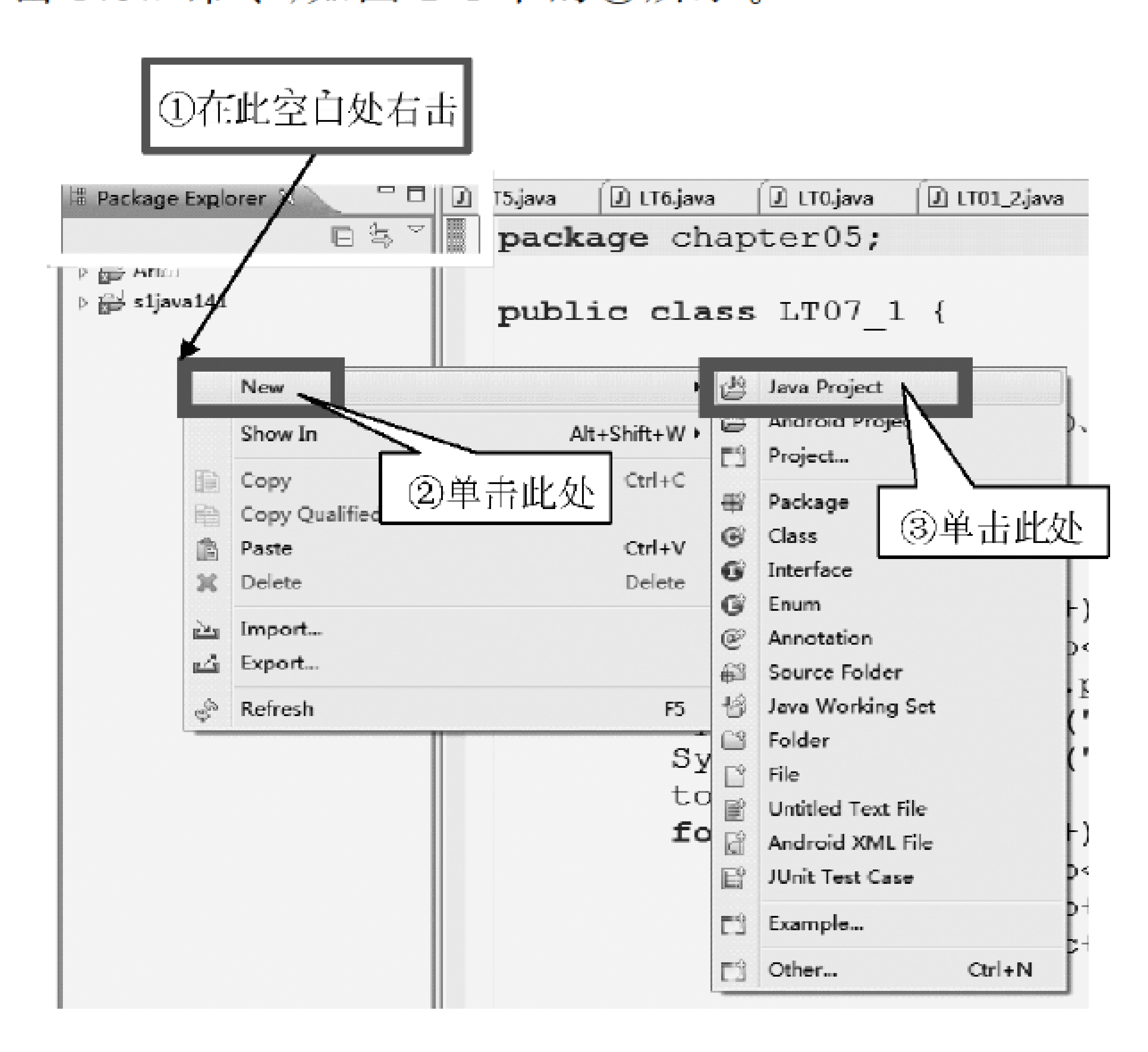


图 1-4　例 1-1 步骤(1)示意图

（2）在 New Java Project 对话框中的 Project name 文本框中输入“ajava”，单击 Finish 按钮，如图 1-5 所示。

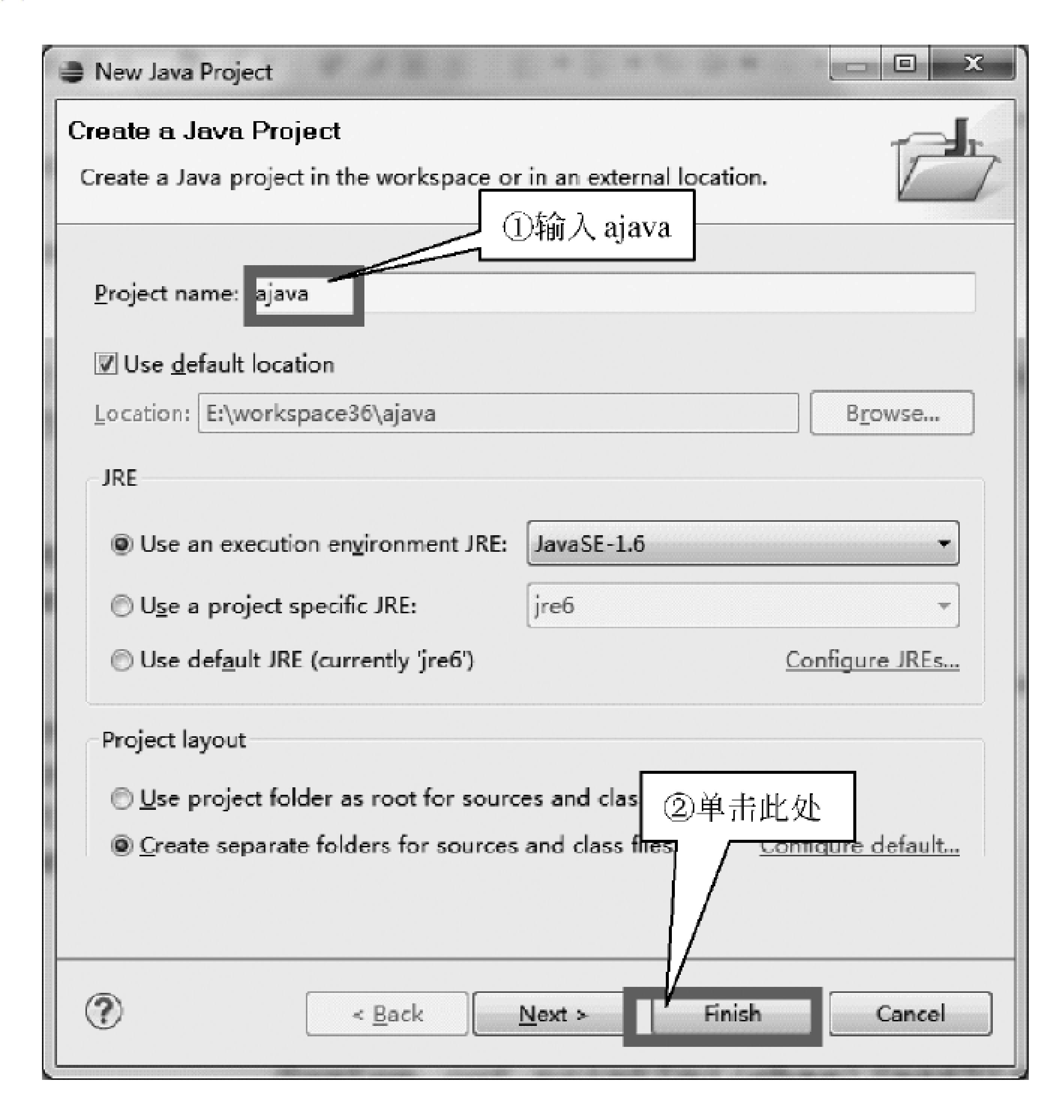


图 1-5　例 1-1 步骤(2)示意图

（3）要在 src 处创建一个名为 com. axt. ch01 的包，按图 1-6 所示步骤操作。

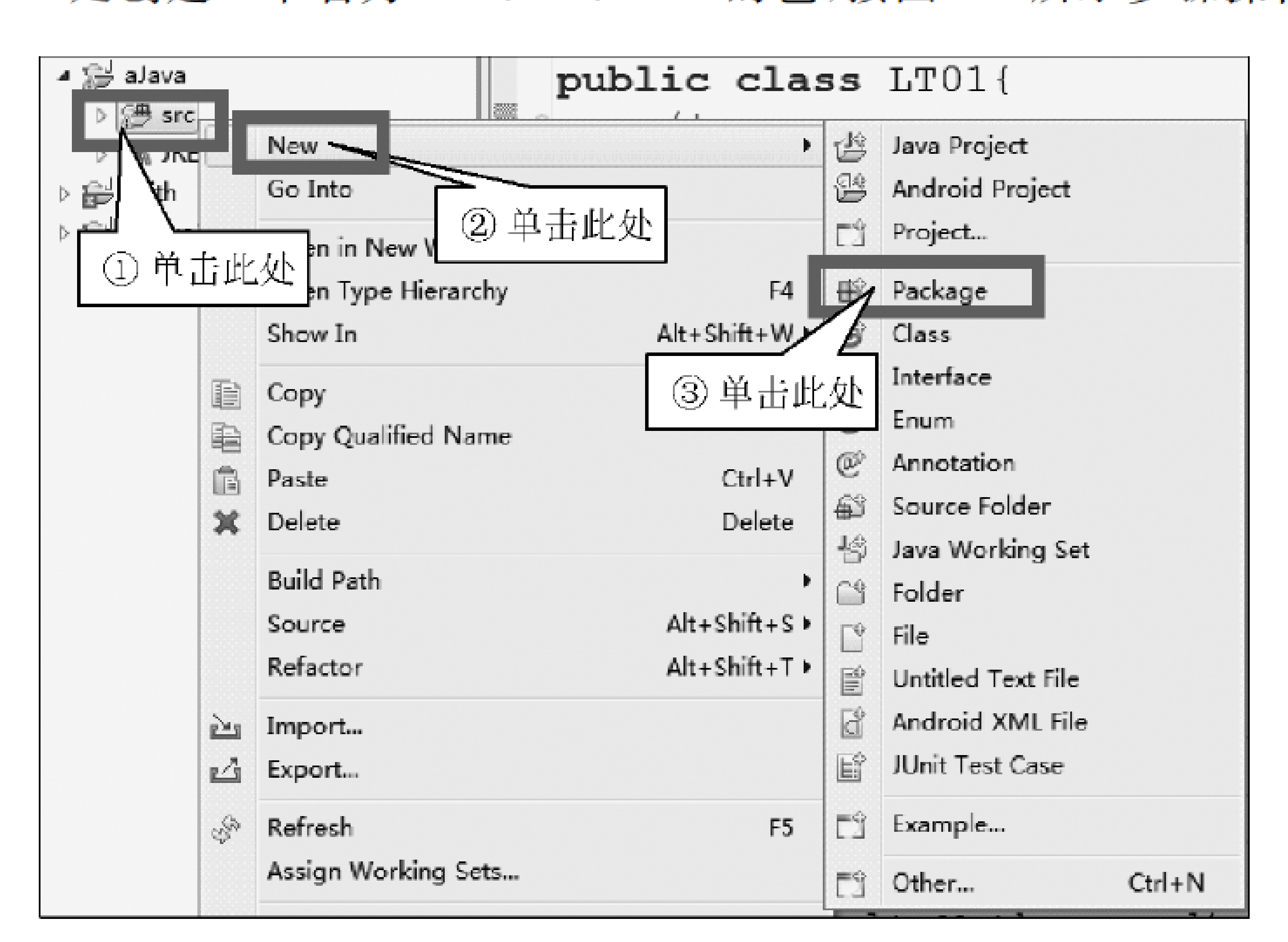


图 1-6　例 1-1 步骤(3)示意图

（4）在 New Java Package 对话框中的 Name 文本框中输入包名“com. axt. ch01”，单击 Finish 按钮，如图 1-7 所示。

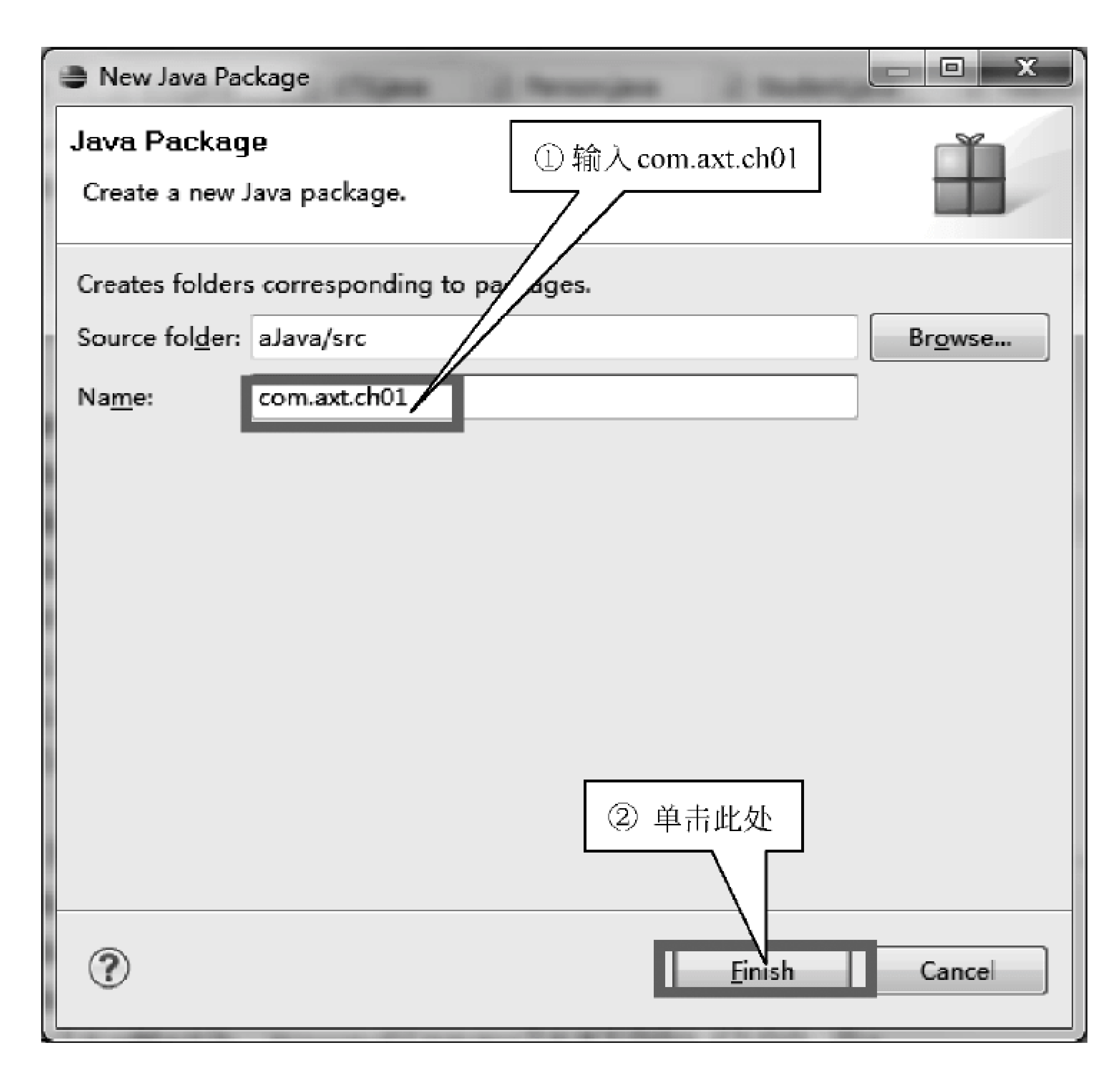


图 1-7 例 1-1 步骤(4)示意图

(5) 在 ch01 包中新建一个 java 源程序,按图 1-8 所示步骤操作。

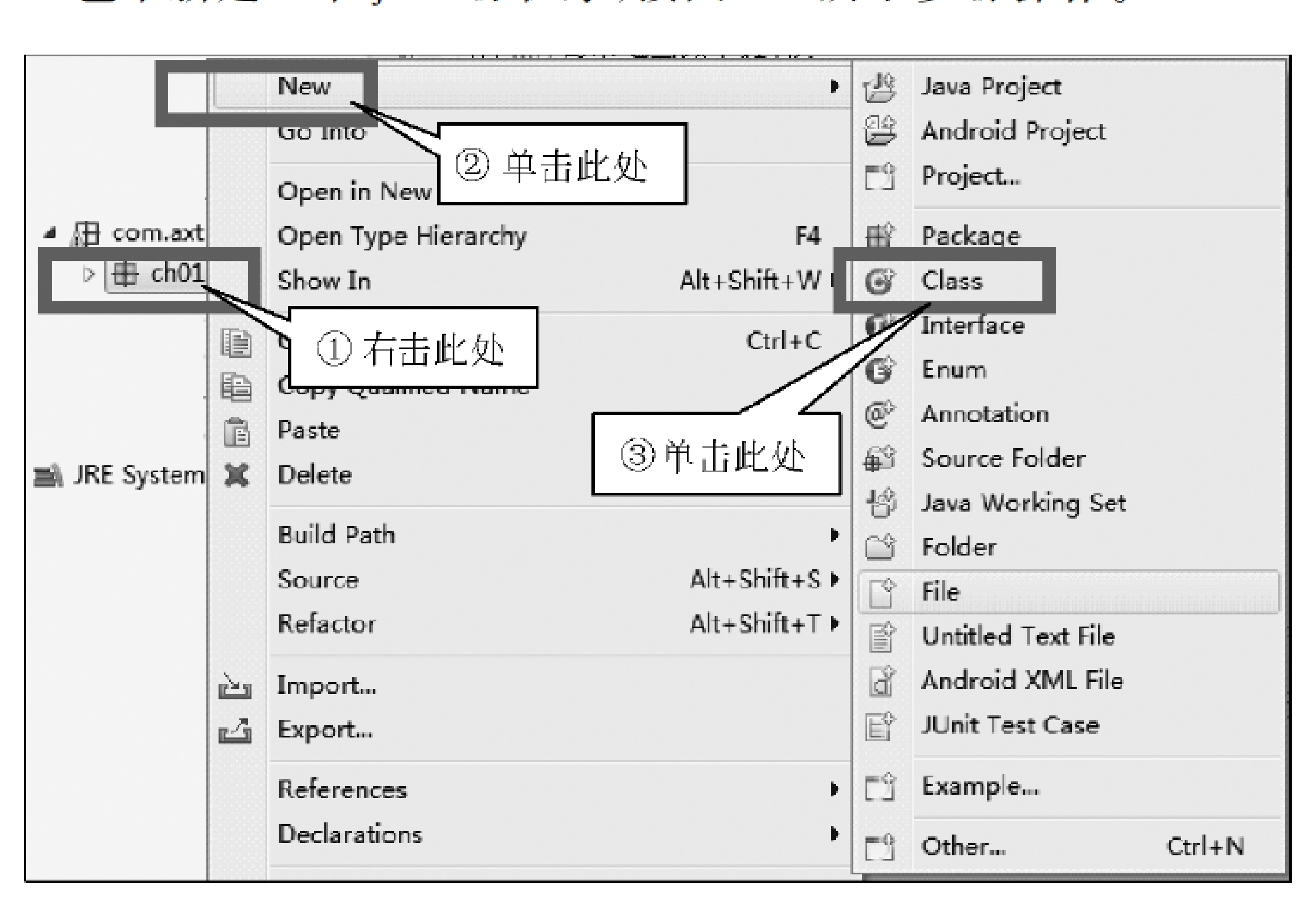


图 1-8 例 1-1 步骤(5)示意图

(6) 在 New Java Class 对话框中的 Name 文本框中输入文件名"LT01. java",如图 1-9 所示,选中 Public static void main 复选框,然后单击 Finish 按钮。

注意:文件名的大小写严格按照提示输入,不要有差错,否则会影响程序的正常运行。

(7) 这时的 Java 程序如图 1-10 中的代码编辑区所示。在声明包名处,package 是声明

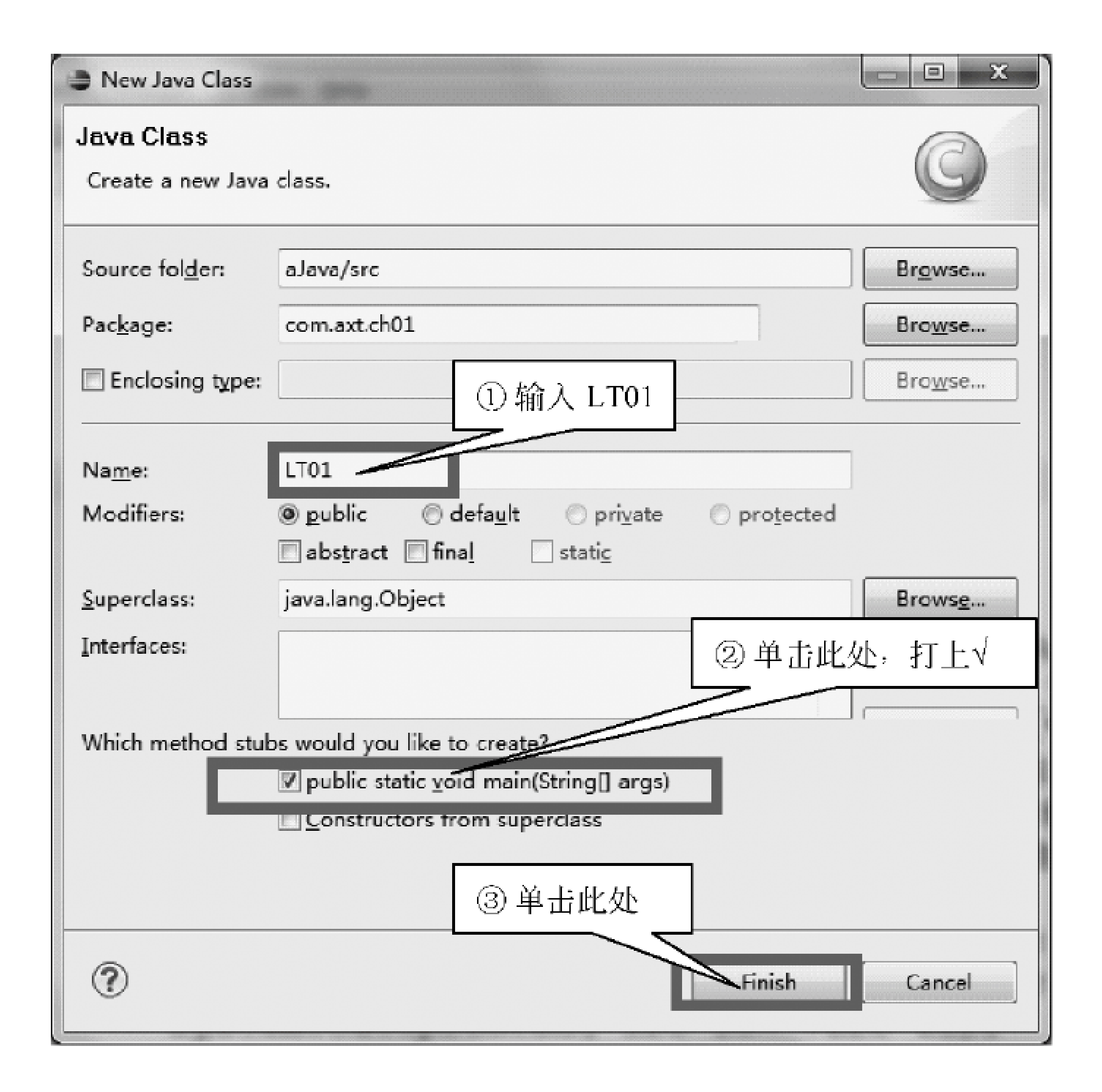


图 1-9　例 1-1 步骤(6)示意图

包的关键字，com. tarena. ch01 是包名；在外层框架处，public class 后面的 LT01 是文件名，行尾的大括号与图 1-10 代码编辑区中最后一行的大括号相匹配，缺一不可；/ ** 与 * /之间为注释语句；main 方法的行尾大括号与倒数第二行的大括号匹配，缺一不可；在代码输入区域按图 1-10 所示输入代码。

(8) 按 F11 键运行程序，在控制台中将会显示：嗨，大家好！

3. Java 项目的结构

Java 程序的运行机制：首先编辑 Java 源程序，Java 源程序是扩展名为. java 的文本文件；其次 Javac. exe(Java 的编译程序)将源程序编译为. class 的字节码文件；最后 Java. exe 程序将字节码文件解释运行出结果。

在 Eclipse 中，通过按 F11 键可将编译源程序为字节码和运行字节码文件一气呵成地完成。Eclipse 将 Java 的源程序放在 src 文件夹下，并按“包”有层次地存放、管理；Java 的字节码文件存放在 bin 文件夹下。

```
package com.axt.ch01; →声明包名。
public class LT01{ →外层框架
    /**
     *这是我的第一个 JAVA程序→注释语句
    */
    public static void main(String[] args){  →main 方法
        main 里的一整块为代码输入区域
    }
}
```

图 1-10　例 1-1 步骤(7)、(8)示意图

例 1-2　按下面的提示操作，查看 LT01. java 和 LT01. class 文件，并进行验证。

(1) 在 Eclipse 集成开发环境界面中，在左侧窗口设置导航器(Navigator)视图，在该视图下可以清楚地看到扩展名为. java 的源程序和扩展名为. class 的字节码文件。按如图 1-11 所示操作。

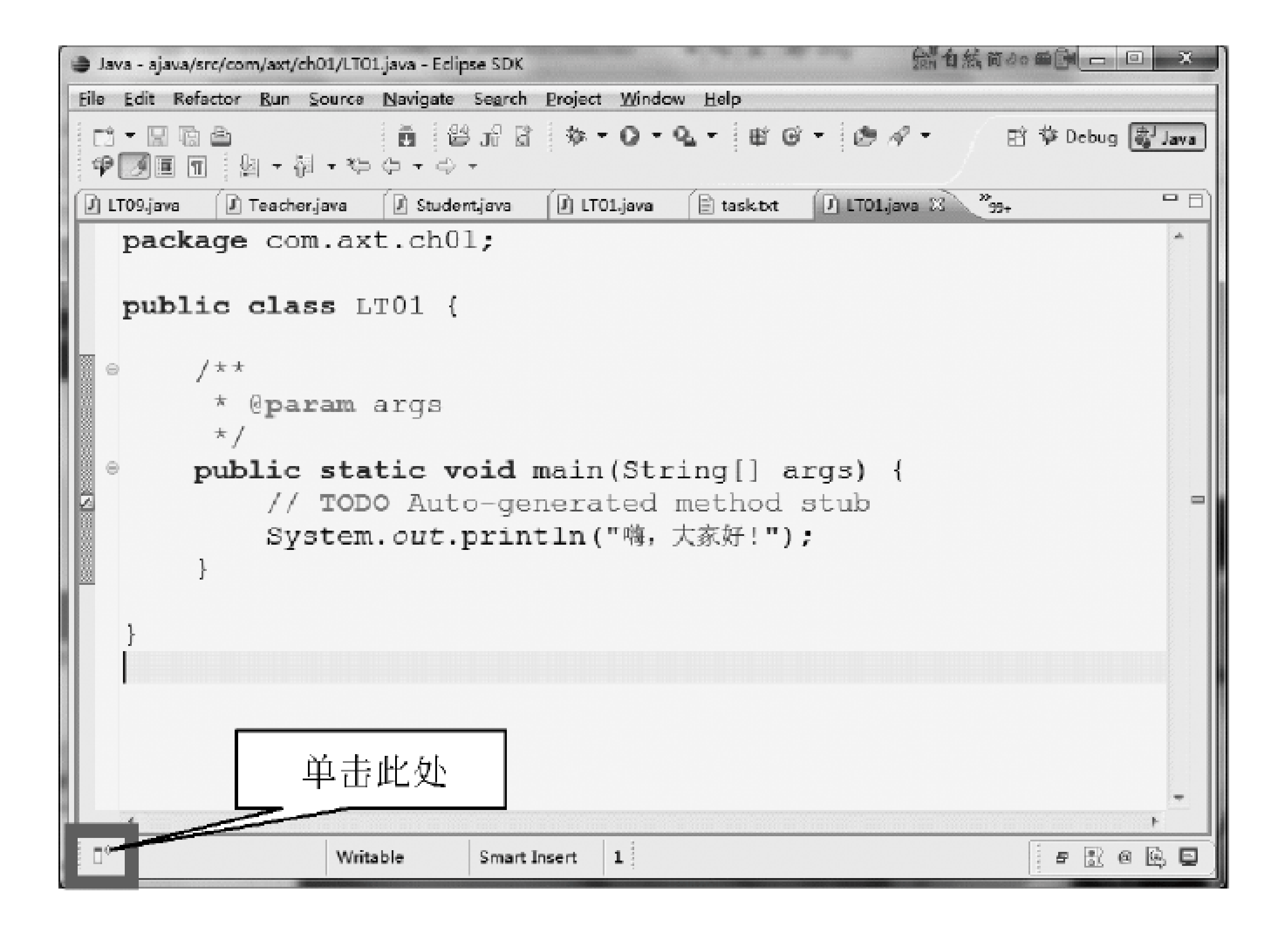


图 1-11　例 1-2 步骤(1)示意图

(2) 接着按图 1-12 所示操作,打开导航器(Navigator)视图。

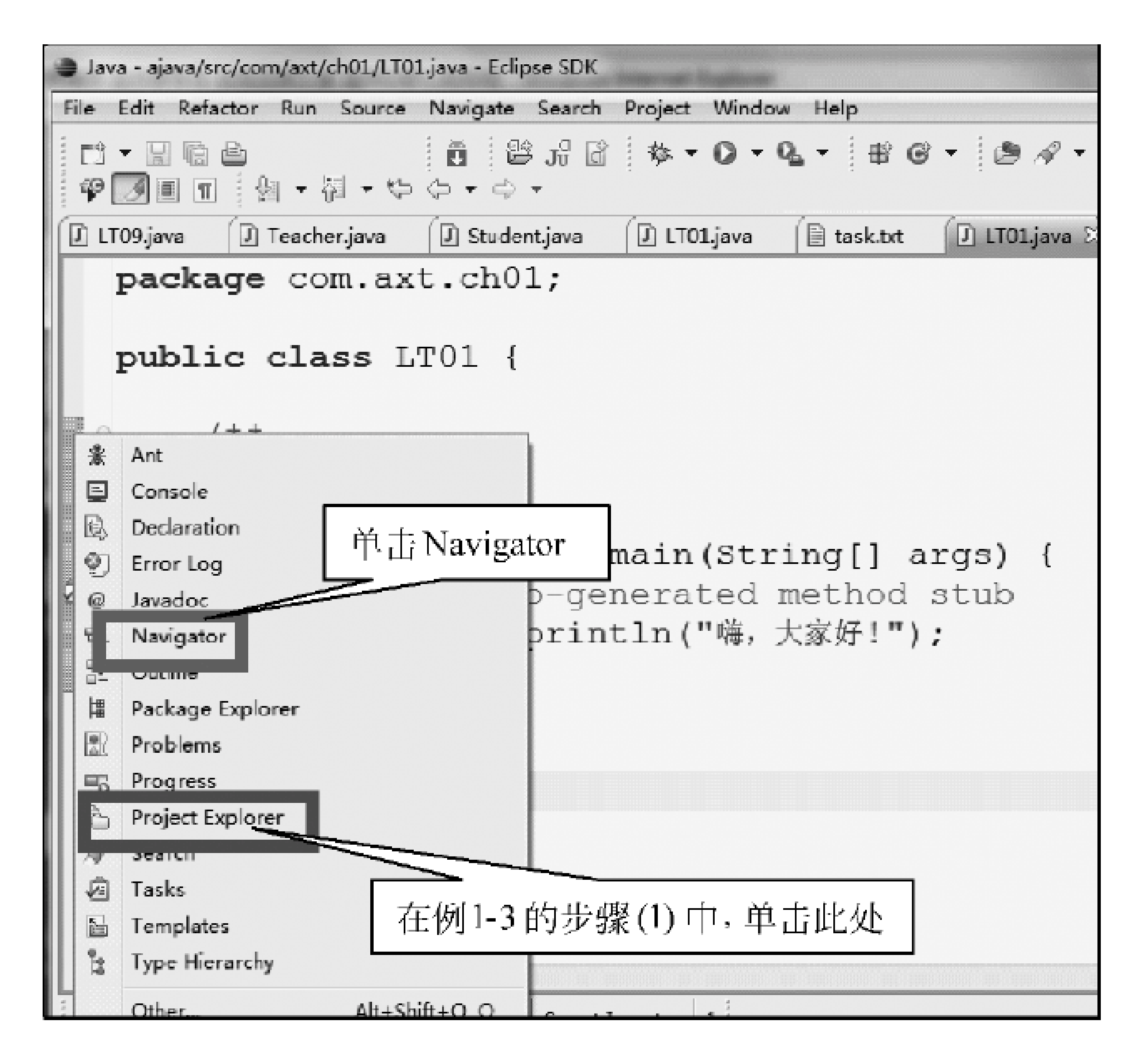


图 1-12　例 1-2 步骤(2)示意图

(3) 按图 1-13 中的①所示操作,可以看到 LT01. class 文件;按图 1-13 中的②所示操作,可以看到 LT01. java 文件。

另外,单击图 1-13 中的③所指向的按钮,是编译、运行当前 Java 程序的又一种方式,与 F11 键的作用相同。

4. 项目的导入、导出

(1) 导出项目:在 Eclipse 中编写的 Java 程序,要用导出的方式将整个项目保存至指定磁盘的文件夹,以便在不同的计算机系统上开发 Java 程序。

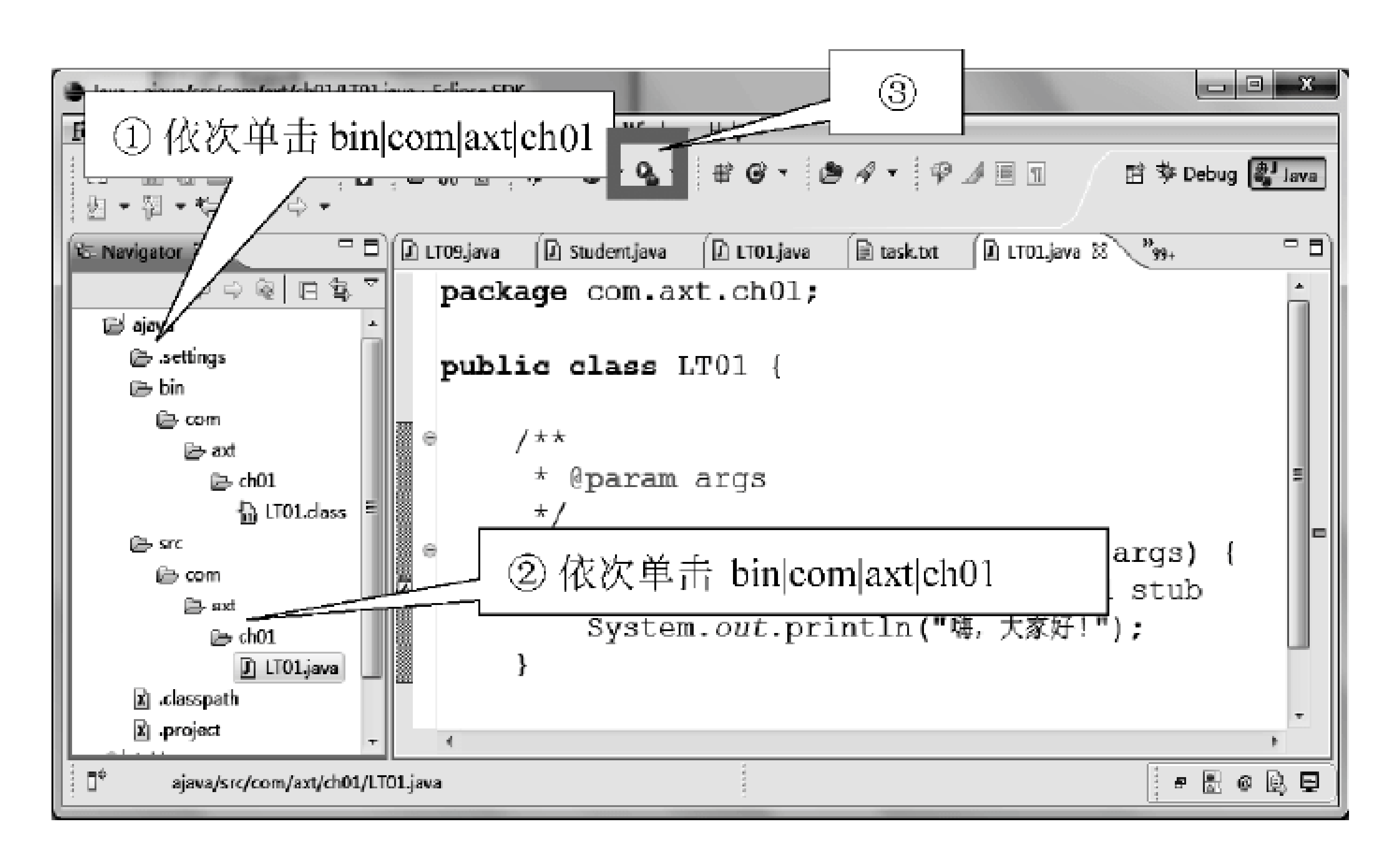


图 1-13　例 1-2 步骤(3)示意图

(2) 导入项目：这是导出项目的逆过程，将外部的 Java 项目导入 Eclipse 中，进行再开发。

例 1-3　按以下步骤将 ajava 项目导出至 E 盘的根目录，导出文件为 ajava. zip。

(1) 按图 1-12 所示打开 Project Explorer(项目资源管理器)，然后按图 1-14 所示操作。

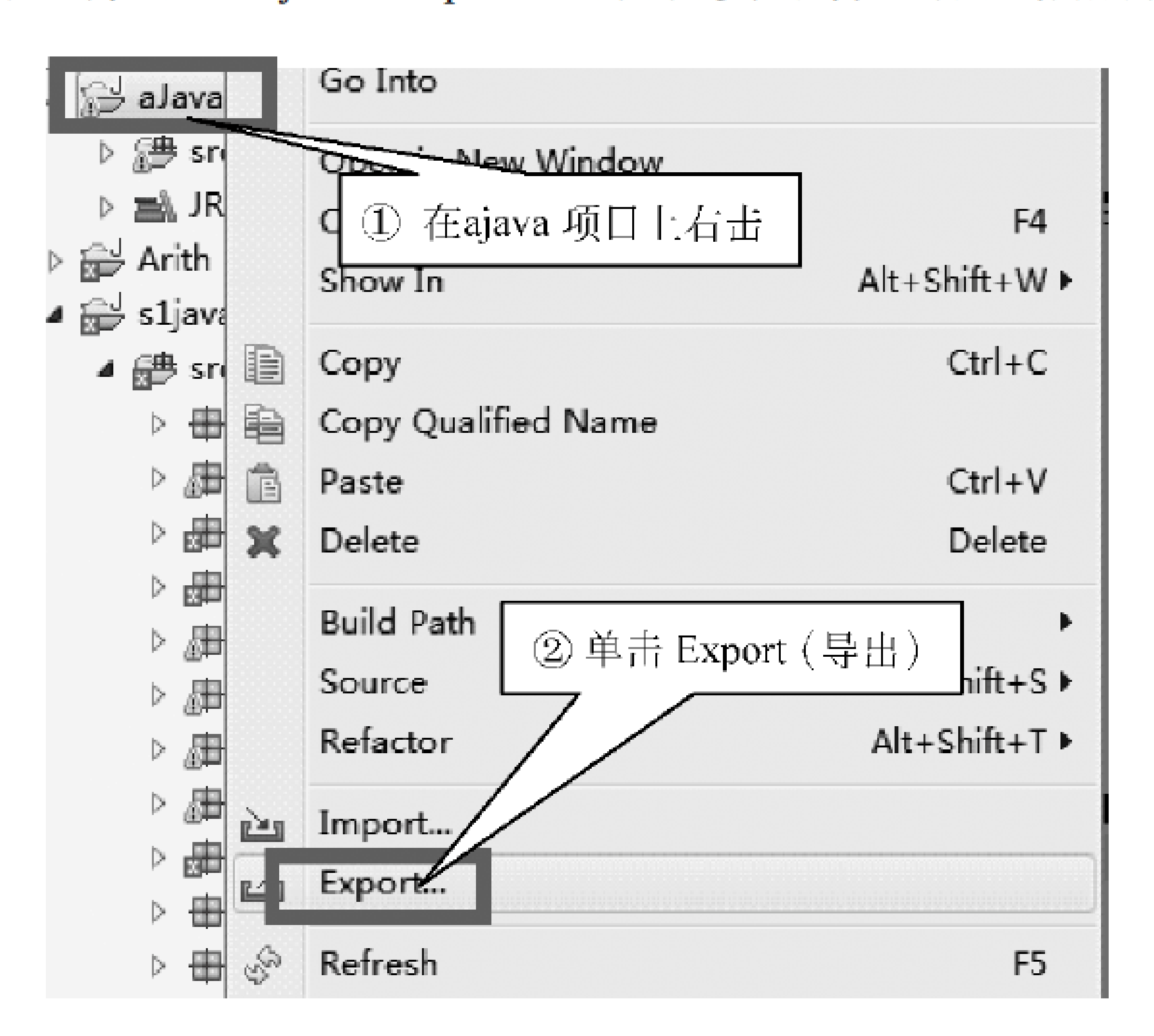


图 1-14　例 1-3 步骤(1)示意图

(2) 选择目标文件的格式，这里选择 Archive File，即压缩文档格式，则导出的项目文件是 zip 压缩文件格式；单击 Next 按钮，如图 1-15 所示。

(3) 选择源项目(默认 ajava 已被选择)，单击 Browse 按钮，设置导出文件的位置和文件名，如图 1-16 所示。

(4) 按图 1-17 所示设置导出文件的位置、文件名。

(5) 在图 1-16 中单击 Finish 按钮，完成导出操作。



图 1-15　例 1-3 步骤(2)示意图

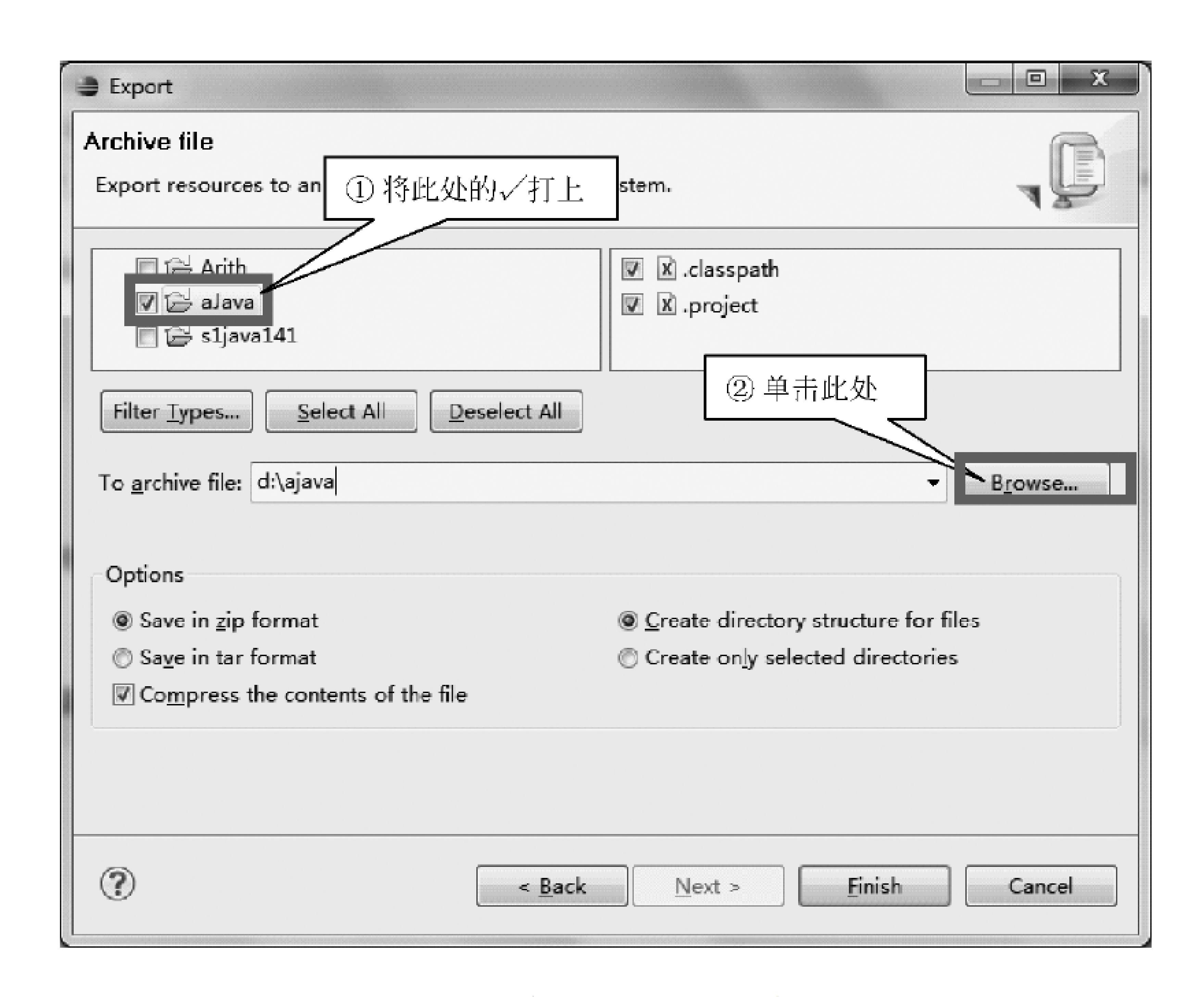


图 1-16　例 1-3 步骤(3)示意图

例 1-4　按以下步骤导入项目 ajava 至 Eclipse，该项目在 E 盘的根目录下。

(1) 先将 Eclipse 中已存在的项目删除，然后才能导入该项目。请在确保 ajava 项目已经成功导出后，完成以下操作将 ajava 项目从 Eclipse 中删除。按图 1-18 所示操作。

(2) 按图 1-19 所示操作。选中 Delete project contents on disk 复选框是将该项目从磁盘的工作空间中删除，若不选中该复选框则下次 ajava 项目将无法导入。

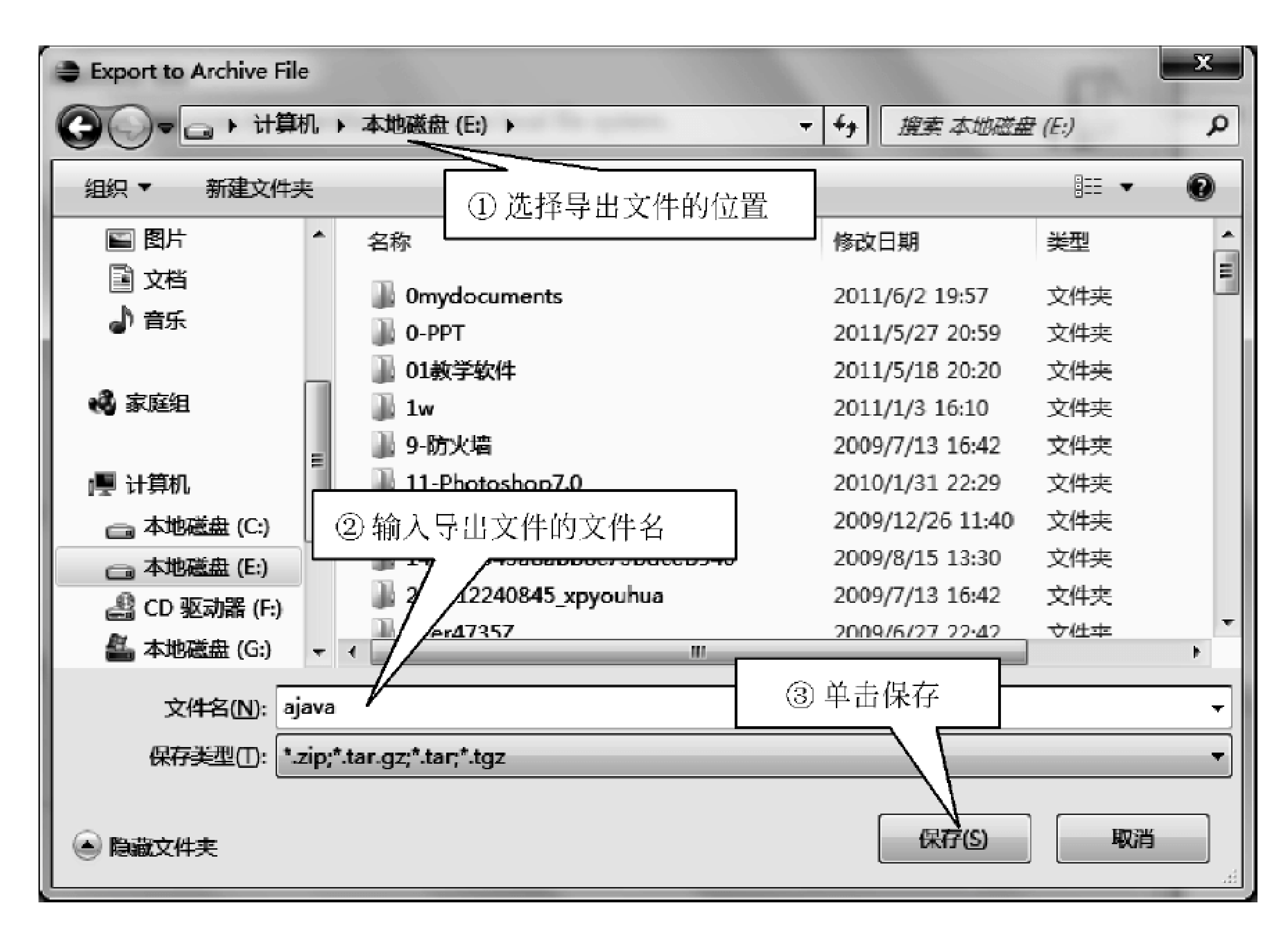


图 1-17　例 1-3 步骤(4)示意图

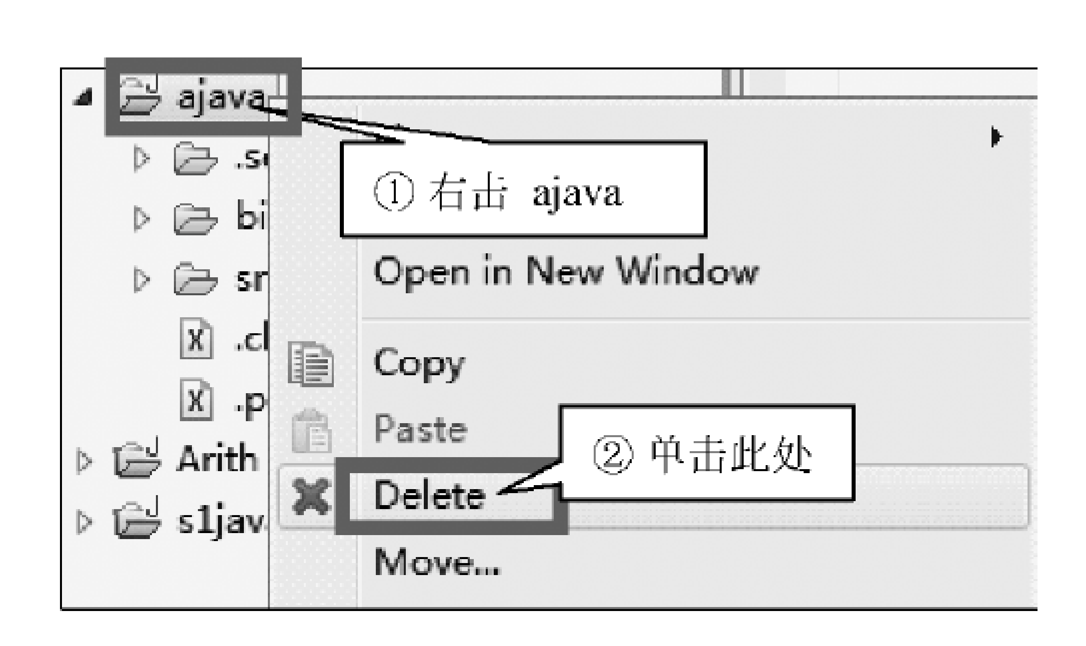


图 1-18　例 1-4 步骤(1)示意图

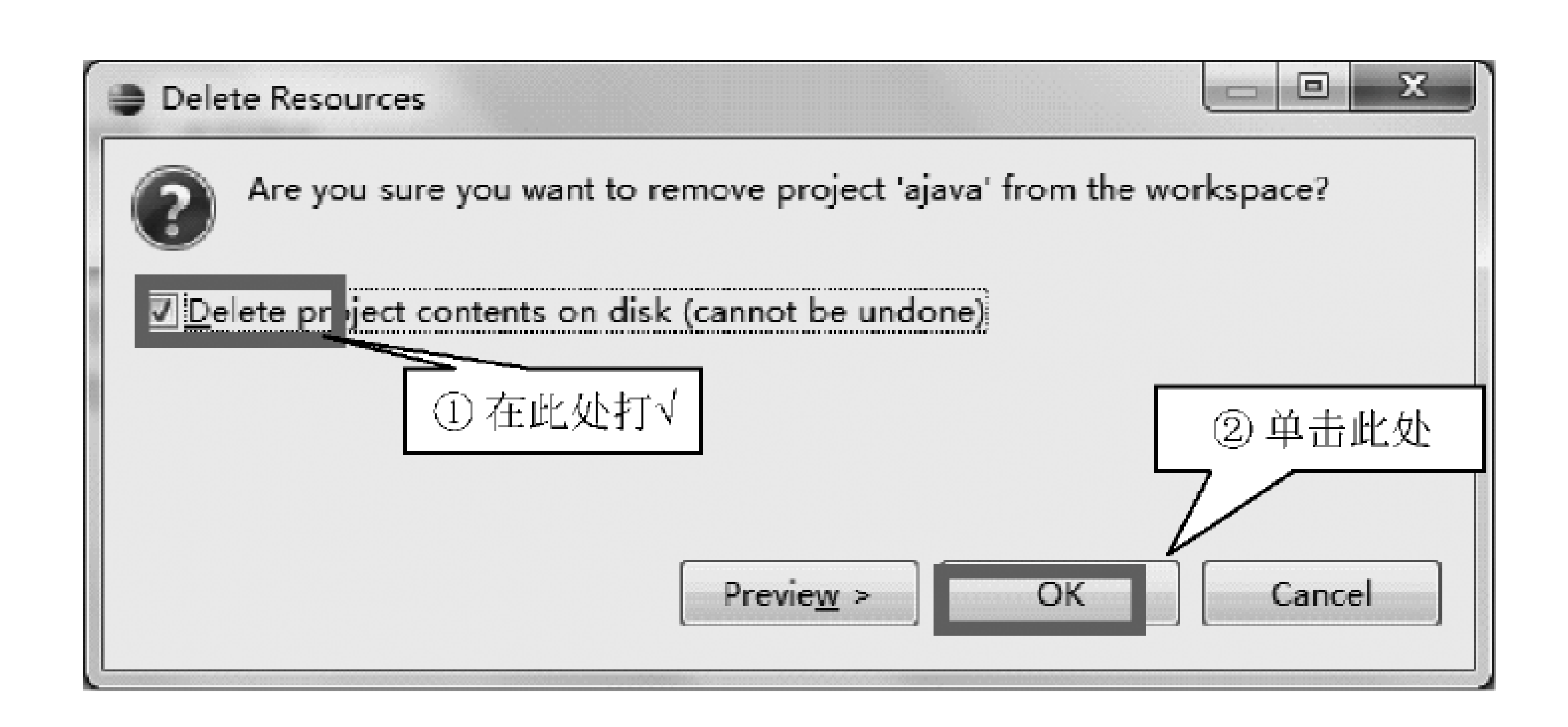


图 1-19　例 1-4 步骤(2)示意图

（3）按图 1-20 所示操作。

（4）按图 1-21 所示操作。图中①所示操作是将已存在的项目导入 Workspace（工作空间）。

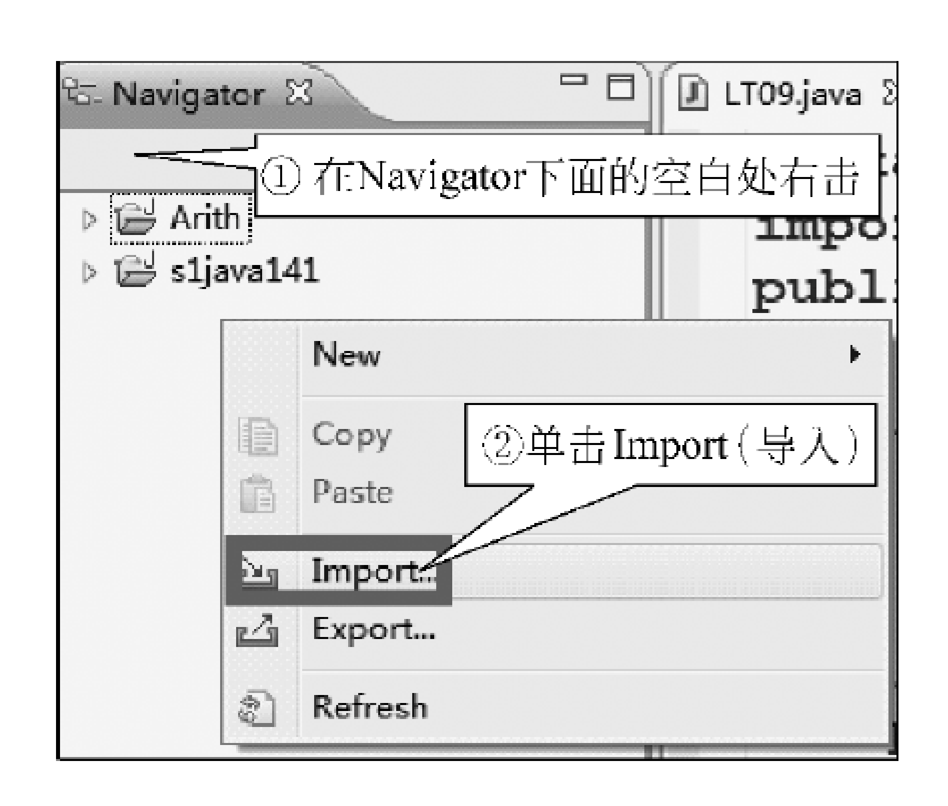


图 1-20　例 1-4 步骤(3)示意图



图 1-21　例 1-4 步骤(4)示意图

（5）按图 1-22 所示操作。

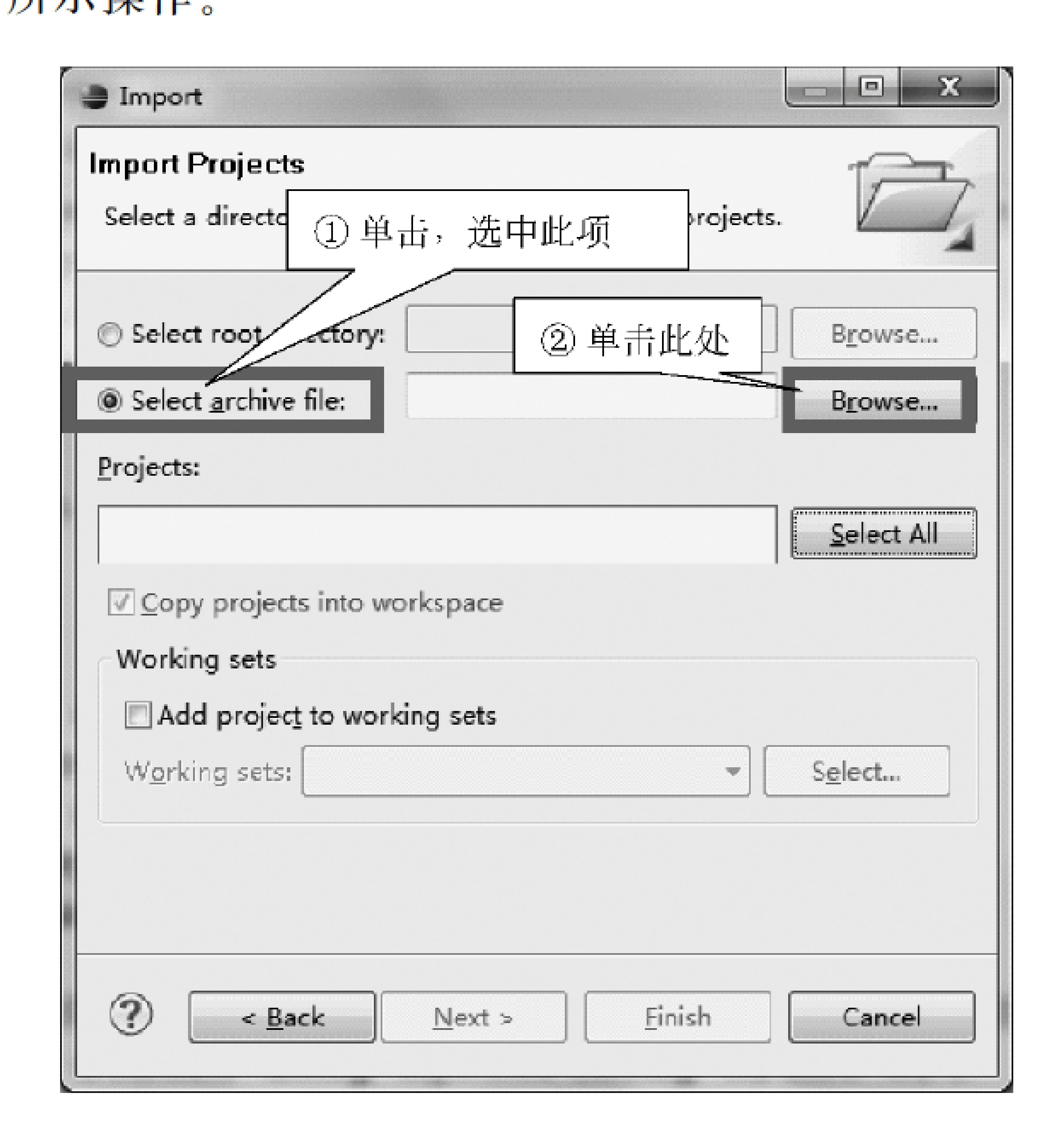


图 1-22　例 1-4 步骤(5)示意图

(6) 按图 1-23 所示操作，选择 E 盘根目录下的 ajava.zip 压缩文件。



图 1-23　例 1-4 步骤(6)示意图

(7) 按图 1-24 所示操作。

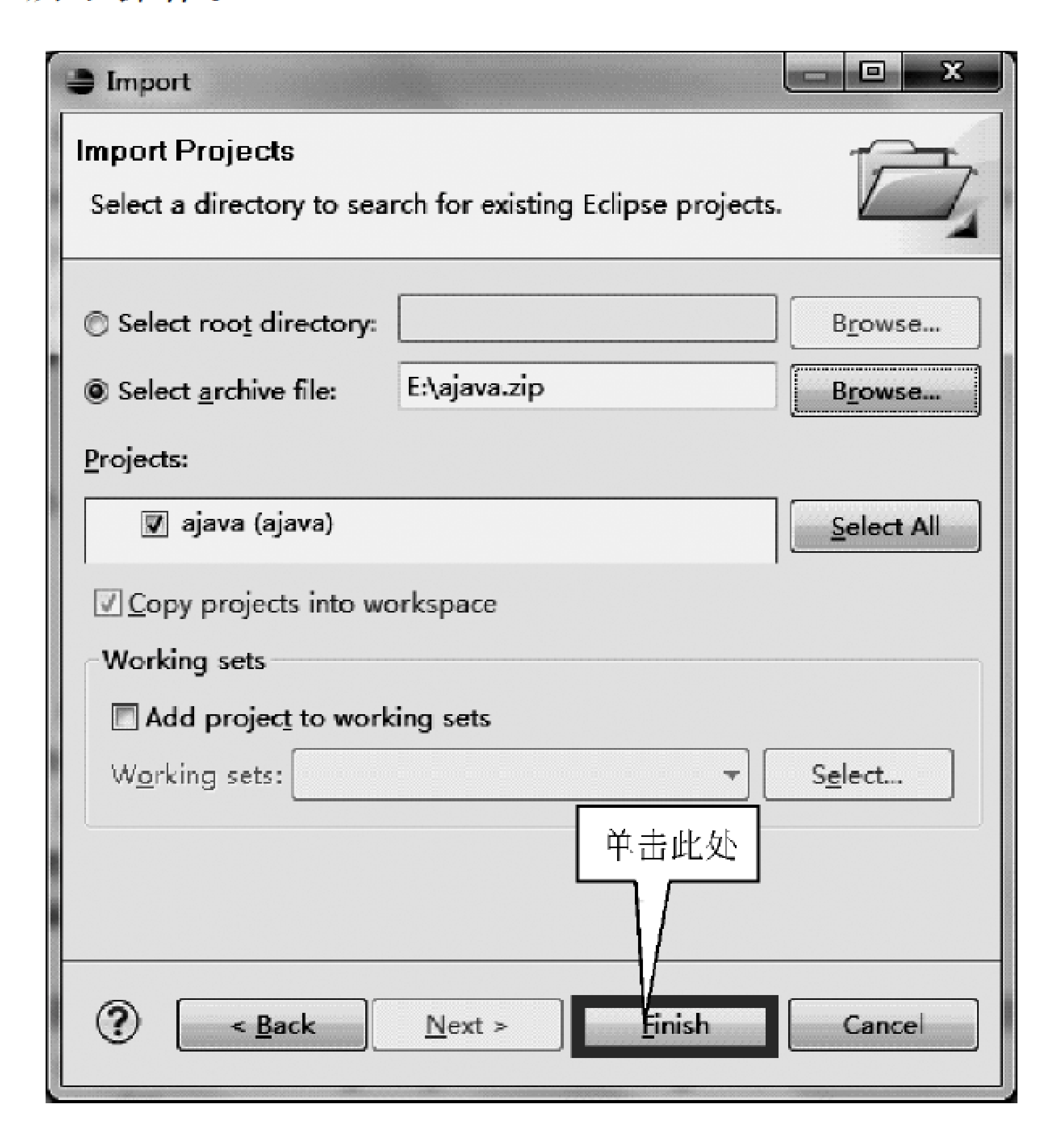


图 1-24　例 1-4 步骤(7)示意图

5. 利用重构修改项目、文件名

项目或程序的名称有时因命名错误等情况需要修改，以下演示如何修改 Java 程序名称(修改项目名称、包(Package)名称的方法相同)。

例 1-5 将文件名称 HelloJava. java 修改为 LT01. java。

(1) 执行例 1-2 中的步骤(2),选择 Package Explorer(包资源管理器),该管理器也是常用的管理 Java 项目的视图。

(2) 按图 1-25 所示操作。其中:

① 左边的视图必须是包资源管理器(Package Explorer)。

② Refactor:重构,快捷键为 Alt+Shift+T 组合键。

③ Rename:重命名,快捷键为 Alt+Shift+R 组合键或 F2 键。

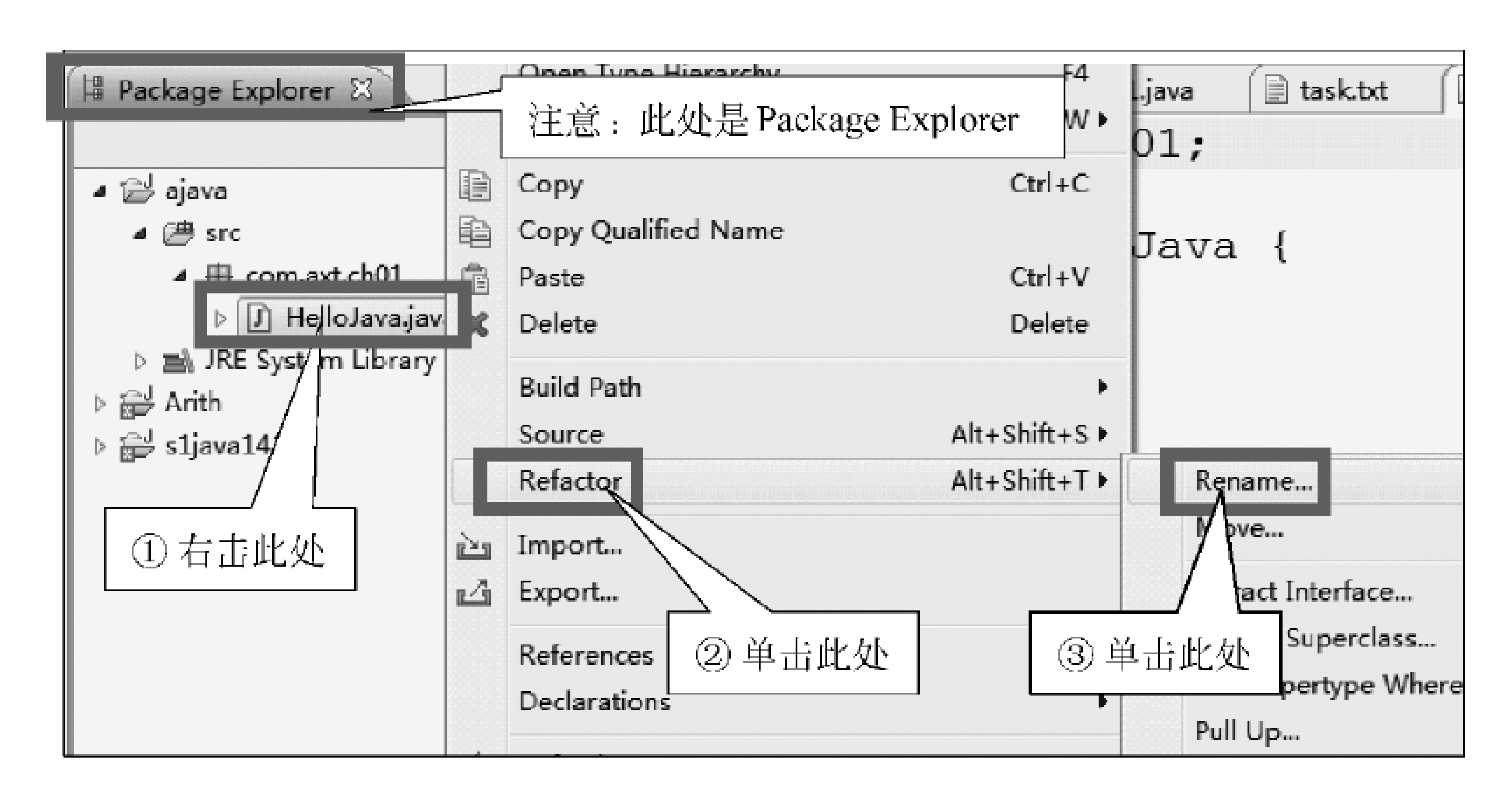


图 1-25 例 1-5 步骤(2)示意图

(3) 按图 1-26 所示操作。

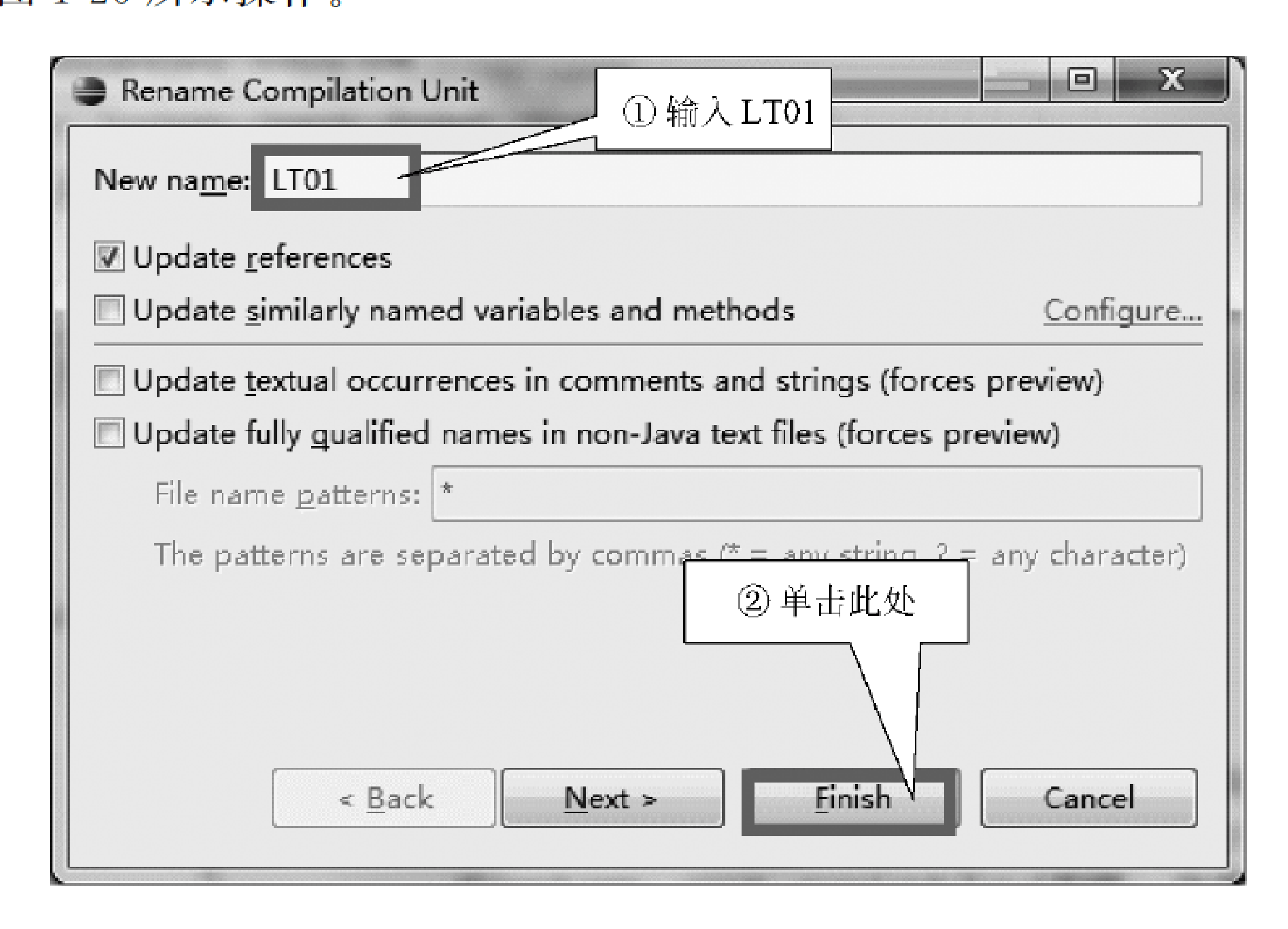


图 1-26 例 1-5 步骤(3)示意图

(4) 按图 1-27 所示操作。

6. Java 的 Math 类(数学运算函数类)

(1) 乘方运算:Math. pow(底数,指数)。



图 1-27　例 1-5 步骤(4)示意图

(2) 开方运算：Math. sqrt(被开方数)。

(3) 取绝对值：Math. abs(被操作数)。

(4) 四舍五入：Math. round(被操作数)，对小数点后第一位四舍五入。

(5) 取整：Math. floor(被操作数)，将小数部分去掉，保留整数部分。

第 2 章　Android 系统概述

本章主要介绍 Android 系统的基本概念以及相关背景及基础知识。通过本章学习，读者应该掌握以下内容：

(1) Android 系统的定义。

(2) Android 系统平台结构。

(3) Android 系统开发过程，主要包括 Android 源码的编译、Android 应用程序模块的应用、创建一个 Hello Android 项目、将界面实现用 XML 编排和调试项目 5 个方面的内容。

(4) Android 开发者联盟、参加 Android 开发者大赛、Android Market。

2.1　Android 系统及背景知识

2.1.1　Android 系统的概念

1. Android 的定义

Android 一词的英文原意是机器人，这里指的是 Google 在 2007 年 11 月发布的基于 Linux 平台的开源智能手机操作系统名称。该平台由操作系统、中间件、用户界面和应用软件组成，是首个为移动终端打造的真正开放和完整的移动软件，主要应用于便携设备。目前尚未有统一中文名称，我国较多人称之为安卓。Android 操作系统最初由 Andy Rubin 开发，最初主要支持手机。2005 年 Google 收购注资，并组建开放手机联盟开发改良，逐渐扩展到平板电脑及其他领域上。Android 的主要竞争对手是苹果公司的 IOS 以及 RIM 的 Blackberry OS。

2. Android 的主要特点

1) 开放性

Android 平台的特点首先就是其开放性，开放的平台允许任何移动终端厂商加入到 Android 联盟中来。显著的开放性可以使其拥有更多的开发者，随着用户和应用的日益丰富，一个崭新的平台也将很快走向成熟。开放性对于 Android 的发展而言，有利于积累人气，这里的人气包括消费者和厂商，而对于消费者来讲，最大的收益正是丰富的软件资源。开放的平台也会带来更大竞争，如此一来，消费者将可以用更低的价位购得满意的手机。

开发者在为其开发程序时拥有更大的自由度，突破了 iPhone 等只能添加为数不多的固定软件的枷锁；同时与 Windows Mobile、塞班等厂商不同，Android 操作系统免费向开发人员提供。获得 Android 操作系统的方式如下：

(1) 网站下载：安卓网、机峰网、91 助理等，支持所有.apk 文件，登录→附件中心→下

载→复制入 SD 卡→安装。

(2) 手机下载：登录→附件中心→下载→安装。

2) 挣脱束缚

在过去很长的一段时间，特别是在欧美地区，手机应用往往受到运营商的制约，使用什么功能、接入什么网络几乎都受到运营商的控制。自从 iPhone 上市，用户可以更加方便地连接网络，运营商的制约减少。随着 EDGE、HSDPA 这些 2G 至 3G 移动网络的逐步过渡和提升，手机随意接入网络已成为现实。

3) 丰富的硬件

这一点还是与 Android 平台的开放性相关，由于 Android 的开放性，众多的厂商会推出千奇百怪、功能特色各具的多种产品。功能上的差异和特色却不会影响到数据同步甚至软件的兼容。例如用户从诺基亚塞班系统手机改用苹果 iPhone，同时还可将塞班中优秀的软件带到 iPhone 上使用，联系人等资料更是可以方便地转移。

4) 开发商

Android 平台提供给第三方开发商一个十分宽泛、自由的环境，因此不会受到各种条条框框的阻挠，这将会促使许多新颖别致的软件诞生。但也有其两面性，如何控制血腥、暴力、情色等方面的程序和游戏正是留给 Android 的难题之一。

5) 无缝结合的 Google 应用

如今叱诧互联网的 Google 已经走过 10 多年的历史，从搜索巨人到全面的互联网渗透，Google 服务如地图、邮件、搜索等已经成为连接用户和互联网的重要纽带，而 Android 平台手机将无缝结合这些优秀的 Google 服务。

3. Android 系统的不足

1) 安全和隐私

由于手机与互联网的紧密联系，个人隐私很难得到保护。除了上网过程中经意或不经意留下的个人足迹，Google 这个巨人也时时站在背后洞察一切，因此，互联网的深入发展将会带来新一轮的隐私危机。

2) 运营商仍然能够影响到 Android 手机

在国内市场，不少用户对购得移动定制机不满，感觉所购的手机被人涂画了广告一般。这样的情况在国外市场同样出现。Android 手机的另一发售运营商 Sprint 就在其机型中内置其手机商店程序。

3) 同类机型用户减少

在不少手机论坛都会有针对某一型号的子论坛，对一款手机的使用心得进行交流，并分享软件资源。而对于 Android 平台手机，由于厂商丰富，产品类型多样，这样使用同一款机型的用户越来越少，缺少统一机型的程序强化。

4) 过分依赖开发商，缺少标准配置

在使用 PC 端的 Windows XP 系统的时候，都会内置微软 Windows Media Player 播放器程序，用户也可以选择更多样的播放器，如 RealPlayer 或暴风影音等，但入手开始使用默认的程序同样可以应付多样的需要。在 Android 平台中，由于其开放性，软件更多依赖第三方厂商，例如 Android 系统的 SDK 中就没有内置音乐播放器，全部依赖第三方开发，缺少了产品的统一性。

2.1.2 Android 背景知识

1. Android 的历史

为了更好地了解 Android,有必要对其历史进行一些介绍。谈到 Android,首先需要了解的是“开放手机联盟”,其英文名称为 Open Handset Alliance,是美国 Google 公司于 2007 年 11 月宣布组建的一个全球性的联盟组织。这一联盟将会支持 Google 公司发布的手机操作系统或者应用软件,共同开发名为 Android 的开放源代码的移动操作系统。

这个联盟成员已经达到了几十家,除了前面提到的 Google 公司外,还包括手机制造商、手机芯片厂商和移动运营商几类。

2. Android 的发展

从开放手机联盟成立以后,Android 的发展也加快了速度,下面简单介绍 Android 的发展历程。

当初很多人对 Google 仅仅发布一款新的、开源智能手机操作系统感到失望,因为另一家巨头苹果公司刚刚推出了自主品牌的 iPhone 手机且在业绩取得了巨大影响力。3 年之后,Android OS 已经成为全球最具潜力、发展最快的手机平台,每天搭载 Android 的新设备出货量高达数十万。相比之下,Google 自主品牌的 Nexusone 经过短暂亮相后仓促退市。

经过一年的发展,Android OS 开始步入主流市场。HTC 推出了第一款基于 Android 的手机设备 T-Mobile G1,虽然不像 iPhone 那样引起巨大轰动,但它向人们阐释了一款真实的 Android 手机到底是什么样的,且展示出 Android 最大的潜在优势,即扩展应用程序能力。除了正常的盈利分红外,Google 官方还拿出 1000 万美元用于奖励那些对 Android 应用程序作出贡献的开发者。

Android 获得 Verizon 的认可,并将其作为 Mo to Droid 等手机的平台。Droid 手机超高的配置和侧滑全键盘设计吸引了无数关注的目光,市场销售突破 100 万部,成为当时的 Google 旗舰手机。与之同期的还有 HTC Incredible 和 Moto Droid X.。

Google 终于发布了自主品牌的 Nexus One 手机,有人称其为 G-Phone。遗憾的是,Google 没有与四大网络运营商和其他销售平台合作,完全使用自己的渠道进行销售。虽然 Nexus One 的各方性能参数很不错,但市场十分惨淡,74 天共计售出 13.5 万部(Droid 和 iPhone 的成绩均过百万),随即无奈退出市场。

HTC 的 Evo 4G 是首部使用 Sprint 高端 Wi Max 网络的智能手机,高清 4.3 英寸显示屏,800×400 分辨率以及 1GHz 处理器使其成为 2010 年甚至 2011 年的新一代旗舰机。

Google Android 2.2 的发布标志着 Android 平台向商业领域迈出了重要一步。Android 2.2 中提供的智能密码策略和远程擦除、兼容 Exchange 日历、自动更新等功能要比旧版提升一大截。

市场调研机构 Gartner 预测,Android 设备出货量将超过苹果和黑莓,成为仅次于诺基亚塞班的业界第二大平台。Android 手机的成功激发了 Google 进军更多其他硬件平台领域,例如平板电脑、上网本等。

Android 手机操作系统在 2011 年成为国内最大的智能手机平台,占智能手机操作系统市场的份额为 68.4%。尽管苹果 iPhone 4S 和 iPhone 4 拥有较高的人气,但其市场份额却

在10%左右。

Android之所以在国内取得如此大的成功,很大一部分原因是该系统得到了诸如华为、中兴以及联想等本土手机制造商的大力支持;另一方面,手机厂商与运营商一起联手推出更具价格竞争力的定制版设备也是推动该系统快速普及的重要因素。

2.1.3 Android系统平台结构

Android的系统平台与其操作系统一样采用了分层结构,共分为4个层,从高到低分别为应用程序层、应用程序框架层、系统运行库层和Linux内核层。

1. 应用程序层

Android会同一系列核心应用程序包一起发布,该应用程序包包括E-mail客户端、SMS短消息程序、日历、地图、浏览器和联系人管理程序等。所有的应用程序都是使用Java语言编写的。

2. 应用程序框架层

开发人员可以完全访问核心应用程序所使用的API框架。该应用程序的结构设计简化了组件的重用;任何一个应用程序都可以发布其功能模块,并且任何其他应用程序都可以使用该功能模块(不过得遵循框架的安全性限制)。同样,该应用程序重用机制也使用户可以方便地替换程序组件。隐藏在每个应用后面的是一系列的服务和系统,其中包括:丰富而又可扩展的视图(Views),可以用来构建应用程序,它包括列表(lists)、网格(grids)、文本框(text boxes)、按钮(buttons),甚至是可嵌入的Web浏览器;内容提供器(Content Providers)使得应用程序可以访问另一个应用程序的数据(如联系人数据库),或者共享它们自己的数据;资源管理器(Resource Manager)提供非代码资源的访问,如本地字符串、图形和布局文件(layout files);通知管理器(Notification Manager)使得应用程序可以在状态栏中显示自定义的提示信息;活动管理器(Activity Manager)用来管理应用程序生命周期并提供常用的导航回退功能。

3. 系统运行库层

系统运行库主要包括两个部分:程序库和Android运行库。

1) 程序库

Android包含一些C/C++库,这些库能被Android系统中不同的组件使用。它们通过Android应用程序框架为开发者提供服务。以下是一些核心库:

(1) 系统C库:是一个从BSD继承来的标准C系统函数库(libc),专门为基于Embedded Linux的设备定制。

(2) Media Framework(媒体库):基于PacketVideo OpenCORE。该库支持多种常用的音频、视频格式回放和录制,同时支持静态图像文件。编码格式包括MPEG4、H.264、MP3、AAC、AMR、JPG和PNG等。其特点为:建立在PacketVideo OpenCORE平台之上;支持标准的视频、音频格式;支持硬件/软件解码插件。

(3) Surface Manager:对显示子系统的管理,并且为多个应用程序提供了2D和3D图层的无缝融合。

(4) WebKit/LibWebCore(Web浏览引擎):一个最新的Web浏览器引擎,支持Android浏览器和一个可嵌入的Web视图。与iPhone相似,Android采用WebKit浏览器

引擎,具备触摸屏、高级图形显示和上网功能,用户能够在手机上查看电子邮件、搜索网址和观看视频节目等,比 iPhone 等其他手机更强调搜索功能,界面更强大,可以说是一种融入全部 Web 应用的单一平台。其特点是:网页渲染以桌面视图模式完整显示;完全地支持 CSS、JavaScript、DOM、AJAX;支持单栏和自适应视图渲染。

(5) SGL:底层的 2D 图形引擎。

(6) 3D Libraries:基于 OpenGL ES 1.0 APIs 实现。该库可以使用硬件 3D 加速(如果可用),或者使用高度优化的 3D 软加速。

(7) FreeType:位图(Bitmap)和矢量(Vector)字体显示。

(8) SQLite:一个对于所有应用程序可用、功能强劲的轻型关系型数据库引擎。其特点是:轻量级事务数据存储;多数平台数据存储的后端。

2) Android 运行库

每一个 Android 应用程序都在它自己的进程中运行,都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 被设计成一个设备,可以同时高效地运行多个虚拟系统。Dalvik 虚拟机执行.dex 格式的 Dalvik 可执行文件,该格式文件针对小内存使用做了优化。同时虚拟机是基于寄存器的,所有的类都经由 Java 编译器编译,然后通过 SDK 中的 dx 工具转化成.dex 格式由虚拟机执行。

Dalvik 虚拟机依赖于 Linux 内核的一些功能,例如线程机制和底层内存管理机制。

4. Linux 内核层

1) Android 核心库

Android 应用程序使用 Java 语言编写,其大部分 Java 语言基础功能都由 Android 核心库提供,例如基础数据结构、函数、I/O、工具、数据库、网络等库。其中大部分实现来源于 ApacheHarmony 项目,核心库的具体实现位于 libcore 目录中,Java 部分最终会被打包为 core.jar 包,经过安装,最终将被放置在目标文件系统的 system\framework\目录中,当桌面启动时首先加载,作为 Java 程序的一个基础包。libcore 中的 C/C++ 代码被编译为 libjavacore.a 静态库,是 Java 核心库的本地代码。另外,libcore 目录中还包括部分测试用例,用来测试 Java 核心库的基本接口功能实现,在移植 Android 或者其虚拟机时,也可以使用它们来测试 Java 核心库的功能。

(1) 核心库主要实现了以下 Java 基础包:

① Java 标准 API(java 包)。

② Java 扩展 API(javax 包)。

③ 企业和组织提供的 Java 类库(org 包)。

(2) Android 系统 API:是基于 Android 核心库实现的 Android 系统应用框架层的 API,即 Android 系统结构图中从上开始的第二层,也就是应用开发人员所使用的 android 包。Android 的每个包都以 android 开头。

(3) android 包:主要负责向 Android 应用程序开发人员提供可用的 API,其实现位于 framework\base\目录中,其中 framework\base\core\java 包含了大部分 API。还有另外一部分 API 属于 Android 系统库中的扩展库部分,分别位于:

① framework\base\graphics\java\。

② framework\base\media\java\。

③ framework\base\opengl\java\。

④ framework\base\wifi\java\。

⑤ framework\base\location\java\。

……

可以看出，其目录和代码的组织形式都比较类似，各个子目录和文件是根据 java 包的关系来组织的（文件夹的层次结构表示包和子包），文件名称和 Java 类的名称一致。例如，android\app\activity. java 对应 android. app. activity 类。android 包的具体实现会通过 JNI 调用本地 C++代码。

(4) Android 资源包：在 android 包中除了系统 API 之外，还包括了部分资源文件，例如图片、多国语言字符串、布局文件等，它们被统一放置在 framework\base\core\res\目录中，最终会被编译为 framework-res. apk 包，放置在目标文件系统的 system\framework\目录中。

(5) ApiCheck 机制：Android 提供了一个 ApiCheck 工具，用来验证经过编译生成的 SDK 的 API 是否符合标准，即 ApiCheck 机制。Android 对于所有的类和 API 都分为开放式和不开放式两种，开放式是可供 SDK 使用的 API，反之则为不开放式，也就是在源码中看到的加"/ ** } * /"注解的 API，它们都不能被使用。为了最大限度地保持 Android 系统的兼容性，ApiCheck 机制要求 Android 代码中所有的 API 都必须和 API 描述文件一致。这里所说的描述文件是位于 framework\base\api\目录中的. xml 文件。打开其中主要的文件 current. xml 可以看到整个文件的内容都包括在<api>…</api>标签之间，作为 Android 的系统 API。其中还包含一些其他的标签，如表 2-1 所示。

表 2-1　API 包含的标签

标　签	描　述	所包含的值
<api>…</api>	Android 系统 API	
<package>…</package>	包标签	
<class>…</class>	类标签	name, extends, abstract, static, final, deprecated, visibility, …
<constructor>…</constructor>	构造函数标签	name, type, visibility, …
<method>…</method>	方法标签	name, return, native, synchronized, …
<implements>…</implements>	需要实现的接口标签	name
<parameter>…</parameter>	参数标签	name, type
<interface>…</interface>	接口标签	name, abstract, static, deprecated, …
<exception>…</exception>	异常标签	name, type
<field>…</field>	字段标签（变量、常量）	name, type, transient, …

2) Linux 内核

Android 的核心系统服务依赖于 Linux2. 6 内核，如安全性、内存管理、进程管理、网络协议栈和驱动模型。Linux 内核也同时作为硬件和软件栈之间的抽象层。它是一个增强内核版本，除了修改部分 Bug 外，提供了用于支持 Android 平台的设备驱动。其核心驱动主要包括以下内容。

(1) Android Binder：基于 OpenBinder 框架的一个驱动，用于提供 Android 平台的进

程间通信(Inter-Process Communication,IPC)。源代码位于 drivers/staging/android/binder.c。

(2) Android 电源管理(PM):一个基于标准 Linux 电源管理系统的轻量级的 Android 电源管理驱动,针对嵌入式设备做了很多优化。源代码位于 kernel/power/earlysuspend.c、kernel/power/consoleearlysuspend.c、kernel/power/fbearlysuspend.c、kernel/power/wakelock.c、kernel/power/userwakelock.c。

(3) 低内存管理器(Low Memory Killer):相对于 Linux 标准 OOM(Out Of Memory)机制更加灵活,它可以根据需要终止进程来释放需要的内存。源代码位于 drivers/staging/android/lowmemorykiller.c。

(4) 匿名共享内存(ashmem):为进程间提供大块共享内存,同时为内核提供回收和管理这个内存的机制。源代码位于 mm/ashmem.c。

(5) Android PMEM(Physical):PMEM 用于向用户空间提供连续的物理内存区域,DSP 和某些设备只能工作在连续的物理内存上。源代码位于 drivers/misc/pmem.c。

(6) Android Logger:一个轻量级的日志设备,用于抓取 Android 系统的各种日志。源代码位于 drivers/staging/android/logger.c。

(7) Android Alarm:提供了一个定时器用于把设备从睡眠状态唤醒,同时它也提供了一个即使在设备睡眠时也会运行的时钟基准。源代码位于 drivers/rtc/alarm.c。

(8) USB Gadget 驱动:一个基于标准 Linux USB gadget 驱动框架的设备驱动。Android 的 USB 驱动是基于 gaeget 框架的,源代码位于 drivers/usb/gadget/。

(9) Android Ram Console:为了提供调试功能,Android 允许将调试日志信息写入一个被称为 RAM Console 的设备里,它是一个基于 RAM 的 Buffer。源代码位于 drivers/staging/android/ram_console.c。

(10) Android timed device:提供了对设备进行定时控制功能,目前支持 vibrator 和 LED 设备。源代码位于 drivers/staging/android/timed_output.c(timed_gpio.c)。

(11) Yaffs2 文件系统:Android 采用 Yaffs2 作为 MTD NAND Flash 文件系统,源代码位于 fs/yaffs2/目录下。Yaffs2 是一个快速稳定的应用于 NAND 和 NOR Flash 的跨平台嵌入式设备文件系统,同其他 Flash 文件系统相比,Yaffs2 使用更小的内存来保存其运行状态,因此占用内存小;Yaffs2 的垃圾回收非常简单而且快速,因此能达到更好的性能;Yaffs2 在大容量的 NAND Flash 上的性能表现尤为明显,非常适合大容量的 Flash 存储。

Android 内核添加或修改的文件很多,下面列出 Android Emulator 内核的文件:

```
drivers/misc/kernel_debugger.c;
drivers/misc/pmem.c;
drivers/misc/qemutrace/qemu_trace_sysfs.c;
drivers/misc/qemutrace/qemu_trace.c;
drivers/misc/qemutrace/qemu_trace.h;
drivers/misc/uid_stat.c;
drivers/staging/android/lowmemorykiller.c;
drivers/staging/android/logger.c;
drivers/staging/android/timed_output.h;
drivers/staging/android/ram_console.c;
drivers/staging/android/timed_gpio.c;
```

```
drivers/staging/android/logger.h;
drivers/staging/android/binder.h;
drivers/staging/android/binder.c;
drivers/staging/android/timed_output.c;
drivers/staging/android/timed_gpio.h;
drivers/rtc/alarm.c;
drivers/rtc/rtc-goldfish.c;
drivers/net/pppolac.c;
drivers/net/ppp_mppe.c;
drivers/net/pppopns.c;
drivers/video/goldfishfb.c;
drivers/switch/switch_class.c;
drivers/switch/switch_gpio.c;
drivers/char/dcc_tty.c;
drivers/char/goldfish_tty.c;
drivers/watchdog/i6300esb.c;
drivers/input/misc/gpio_event.c;
drivers/input/misc/gpio_input.c;
drivers/input/misc/gpio_output.c;
drivers/input/misc/keychord.c;
drivers/input/misc/gpio_axis.c;
drivers/input/misc/gpio_matrix.c;
drivers/input/keyreset.c;
drivers/input/keyboard/goldfish_events.c;
drivers/input/touchscreen/synaptics_i2c_rmi.c;
drivers/usb/gadget/android.c;
drivers/usb/gadget/f_adb.h;
drivers/usb/gadget/f_mass_storage.h;
drivers/usb/gadget/f_adb.c;
drivers/usb/gadget/f_mass_storage.c;
drivers/mmc/host/goldfish.c;
drivers/power/goldfish_battery.c;
drivers/leds/ledtrig-sleep.c;
drivers/mtd/devices/goldfish_nand_reg.h;
drivers/mtd/devices/goldfish_nand.c;
kernel/power/earlysuspend.c;
kernel/power/consoleearlysuspend.c;
kernel/power/fbearlysuspend.c;
kernel/power/wakelock.c;
kernel/power/userwakelock.c;
kernel/cpuset.c;
kernel/cgroup_debug.c;
kernel/cgroup.c;
mm/ashmem.c;
include/linux/ashmem.h;
include/linux/switch.h;
include/linux/keychord.h;
include/linux/earlysuspend.h;
include/linux/android_aid.h;
include/linux/uid_stat.h;
```

```
include/linux/if_pppolac.h;
include/linux/usb/android.h;
include/linux/wifi_tiwlan.h;
include/linux/android_alarm.h;
include/linux/keyreset.h;
include/linux/synaptics_i2c_rmi.h;
include/linux/android_pmem.h;
include/linux/kernel_debugger.h;
include/linux/gpio_event.h;
include/linux/wakelock.h;
include/linux/if_pppopns.h;
net/ipv4/sysfs_net_ipv4.c;
net/ipv4/af_inet.c;
net/ipv6/af_inet6.c;
net/bluetooth/af_bluetooth.c;
security/commoncap.c;
fs/proc/base.c。
```

2.2 Android 系统开发概述

2.2.1 详解 Android 源码的编译

本节介绍如何设置 Android 源码的编译环境(基于 Android 1.0),包括 Linux 下的配置,旨在对读者了解 Android 开发有所帮助。

本次编译过程主要参考官方文档 http://source.Android.com/download,编译环境为 Ubuntu 8.10。

1. 安装相关环境

1) 安装 Java 环境

```
sudo apt - get install sun - java6 - jre sun - java6 - plugin sun - java6 - fonts sun - java6 - jdk
```

需要说明的是,官方文档说如果用 sun-java6-jdk 出问题时,可用 sun-java5-jdk。经测试发现,如果仅仅 make(make 不包括 make sdk),用 sun-java6-jdk 是没有问题的,而 make sdk 就会有问题,严格来说是在 make doc 出了问题,它需要的 javadoc 版本为 1.5。因此,安装完 sun-java6-jdk 后最好再安装 sun-java5-jdk,或者只安装 sun-java5-jdk。

```
sudo apt - get install sun - java5 - jdk
```

这里 sun-java6-jdk 和 sun-java5-jdk 都安装,并只修改 javadoc.1.gz 和 javadoc 的 link。因为只有这两个文件是 make sdk 用到的。这样的话,除了 javadoc 工具是用 1.5 版本,其他均用 1.6 版本。

2) 修改 javadoc 的 link

```
cd /etc/alternatives
sudo rm javadoc.1.gz
```

```
sudo ln -s /usr/lib/jvm/java-1.5.0-sun/man/man1/javadoc.1.gz javadoc.1.gz
sudo rm javadoc
sudo ln -s /usr/lib/jvm/java-1.5.0-sun/bin/javadoc javadoc
```

2. 设置环境变量

```
vim ~/.bashrc
```

在.bashrc 中新增或整合 PATH 变量，代码如下：

```
#java 程序开发/运行的一些环境变量
JAVA_HOME=/usr/lib/jvm/java-6-sun
JRE_HOME=${JAVA_HOME}/jre
export Android_JAVA_HOME=$JAVA_HOME
export CLASSPATH=.:${JAVA_HOME}/lib:$JRE_HOME/lib:$CLASSPATH
export JAVA_PATH=${JAVA_HOME}/bin:${JRE_HOME}/bin
export JAVA_HOME;
export JRE_HOME;
export CLASSPATH;
HOME_BIN=~/bin/
export PATH=${PATH}:${JAVA_PATH}:${JRE_PATH}:${HOME_BIN};
#echo $PATH;
```

最后，同步这些变化：

```
source ~/.bashrc
```

3. 安装 repo(用来更新 Android 源码)

(1) 创建~/bin 目录，用来存放 repo 程序，命令如下：

```
$ cd ~
$ mkdir bin
```

并加到环境变量 PATH 中。

(2) 下载 repo 脚本并使其可执行：

```
$ curl http://Android.git.kernel.org/repo >~/bin/repo
$ chmod a+x ~/bin/repo
```

4. 下载 Android 源码并更新

建议不要用 repo 来下载(Android 源码超过 1GB，下载速度会非常慢)，直接在网上下载，地址为 http://www.Androidin.com/bbs/pub/cupcake.tar.gz，解压出来的 cupcake 目录下也有 repo 文件夹，可以通过 repo sync 来更新 cupcake 代码：

```
tar -xvf cupcake.tar.gz
```

更新很慢，用了大约 3 个小时。

5. 编译 Android 源码并得到～/project/Android/cupcake/out 目录

进入 Android 源码目录后，执行命令：

```
make
```

这一过程约 2 个小时。

6. 在模拟器上运行编译好的 Android 源码

Android SDK 的 Emulator(模拟器)程序在 Android-sdk-linux_x86-1.0_r2/tools/目录下。Emulator 是需要加载一些 image 的，默认加载 Android-sdk-linux_x86-1.0_r2/tools/lib/images 目录下的 kernel-qemu(内核) ramdisk.img、system.img、userdata.img。

(1) 编译好 Android 之后，Emulator 在～/project/Android/cupcake/out/host/linux-x86/bin 目录下，ramdisk.img、system.img、userdata.img，～/project/Android/cupcake/out/target/product/generic 目录下。

(2) 进入 emulator 所在目录：

```
cd ～/project/Android/cupcake/out/host/linux - x86/bin
```

增加环境变量：

```
vim ～/.bashrc
```

(3) 在.bashrc 中新增环境变量，代码如下：

```
# java 程序开发/运行的一些环境变量
export Android_PRODUCT_OUT = ～/project/Android/cupcake2/out/target/product/generic
Android_PRODUCT_OUT_BIN = ～/project/Android/cupcake2/out/host/linux - x86/bin
export PATH = ${PATH}:${Android_PRODUCT_OUT_BIN};
```

(4) 同步这些变化：

```
source ～/.bashrc
emulator - image system.img - data userdata.img - ramdisk ramdisk.img
```

(5) 进入 Android 桌面，就说明编辑 Android 源码成功了。

在 out/host/linux-x86/bin 目录下生成许多有用工具，包括 Android SDK/tools 的所有工具，因此可以把 Eclipse 中 Android SDK 的路径指定到 out/host/linux-x86/bin 目录进行开发。

7. 编译 Linux Kernel

直接 make Android 源码时，并没有 make Linux Kernel，因此在运行 Emulator 时不用编译 Linux Kernel。如果要移植 Android 或增删驱动，则需要编译 Linux Kernel。Linux Kernel 的编译将在后文中介绍。

8. 编译模块

Android 中的一个应用程序可以单独编译，编译后要重新生成 system.img。

在源码目录下执行命令：

```
. build/envsetup.sh # 注意 build 与其前面的“”之间有空格
```

执行该命令后就多出一些命令如下：

```
- croot:Changes directory to the top of the tree.      //更改目录树的顶部
- m:Makes from the top of the tree.     //从树的顶部生成
- mm: Builds all of the modules in the current directory.      //在当前目录下生成所有模块
- mmm: Builds all of the modules in the supplied directories.   //在提供的目录中构建所有模块
- cgrep: Greps on all local C/C++files.      //在所有本地 C/C++文件里查找字符串
- jgrep: Greps on all local Java files.      //在所有本地 java 文件里查找字符串
- resgrep: Greps on all local res/ * .xml files.      //在所有本地根目录下查找字符串
- godir: Go to the directory containing a file.      //跳转至包含文件的目录
```

可以加-help 查看用法。可以使用 mmm 来编译指定目录的模块，例如编译联系人：

```
mmm packages/apps/Contacts/
```

编译完成之后生成如下两个文件：

```
out/target/product/generic/data/app/ContactsTests.apk
out/target/product/generic/system/app/ Contacts.apk
```

可以使用 make snod 命令重新生成 system.img，再运行 Emulator。

9. 编译 SDK

直接执行 make 是不包括 make sdk 的。make sdk 用来生成 SDK，这样就可以用与源码同步的 SDK 来开发 Android 了。

1）修改/frameworks/base/include/utils/Asset.h 文件

将代码

```
UNCOMPRESS_DATA_MAX = 1 * 1024 * 1024
```

改为：

```
UNCOMPRESS_DATA_MAX = 2 * 1024 * 1024
```

原因是 Eclipse 编译工程需要大于 1.3MB 的 Buffer。

2）编译 ADT

ADT 是一个 Eclipse 插件，作用是关联 Android SDK，使得 Eclipse 能够新建 Android 工程。

编译 ADT 的步骤如下：

进入 cupcake 源码的 development/tools/eclipse/scripts 目录，执行命令：

```
export ECLIPSE_HOME = Eclipse 路径
./build_server.sh 想建立 ADT 的路径
```

注意是先执行这一步再执行下面的 make sdk，因为在执行./build_server.sh 时会把生成的 SDK 清除。

3）执行 make sdk

注意这里需要的 javadoc 版本为 1.5。

make sdk 编译很慢。编译后生成的 SDK 存放在 out/host/linux-x86/sdk/目录下，此目录下有 Android-sdk_eng.xxx_linux-x86.zip 和 Android-sdk_eng.xxx_linux-x86 两个目录，后者就是 SDK 目录。

实际上，当用 mmm 命令编译模块时，一样会把 SDK 的输出文件清除，因此最好把 Android-sdk_eng.xxx_linux-x86 目录移出来。

4）安装、配置 ADT

安装、配置 ADT 请参考官方文档。

5）查看 AVD

编译出来的 SDK 是没有 AVD(Android Virtual Device，Android 运行的虚拟设备)的，这可以通过 Android 工具查看。执行命令：

```
Android list
```

输出为：

```
Available Android targets:
[1] Android 1.5
     API level: 3
     Skins: HVGA-P, QVGA-L, HVGA-L, HVGA (default), QVGA-P
Available Android Virtual Devices:
```

表明没有 AVD。如果没有 AVD，则 Eclipse 编译工程时会出错（Failed to find a AVD compatible with target 'Android 1.5'. Launch aborted.）。

6）创建 AVD

```
Android create avd -t 1 -c ~/sdcard.img -n myavd
```

可以用 Android - help 来查看上面命令参数的用法。创建中有一些参数默认就行了。

再执行 Android list 命令可以看到 AVD 存放的位置。以后每次运行 Emulator 都要加-avd myavd 或@myavd 参数，这样 Eclipse 才会在打开的 Emulator 中调试程序。

这样 SDK 和 ADT 就生成了，可以按照官方文档把它们整合到 Eclipse 中，这里不再细述。

读者可以建个 Android 的新工程，试试自己编译的 SDK。

2.2.2 Android 应用程序模块详解

1. 理解 3 个基础技术

在大多数操作系统中存在独立的一对一的可执行文件，例如 Windows 中的 exe 文件，它可以产生进程，并能和界面图标、应用进行用户交互。但在 Android 中，这是不固定的，需

要将这些分散的部分进行组合。由于 Android 应用程序模块这种可灵活变通的特点，在实现一个应用的不同部分时需要理解一些基础技术。

1）一个 android 包（简称 .apk）

android 包里面包含应用程序的代码以及资源。这是一个应用发布，用户能下载并安装其设备上的文件。

2）一个任务

一般情况下用户可以当它为一个应用程序来启动。通常在桌面上会有一个图标用来启动任务，这是一个上层的应用，可以将任务切换到前台来。

3）一个进程

这是一个底层的代码运行级别的核心进程。通常.apk 包里所有代码运行在一个进程里，一个进程对应于一个.apk 包。但是进程标签常用来改变代码运行的位置，可以是全部的.apk 包或者是独立的活动（Activity）、接收器、服务或提供器组件等。

2. 任务

当用户看到的“应用”无论实际是如何处理的，它都是一个任务。如果仅仅通过一些活动来创建一个.apk 包，其中有一个肯定是上层入口（通过动作的 intent-filter 以及分类 android.intent.category.LAUNCHER），然后该.apk 包就创建了一个单独任务，无论启动哪个活动都会是这个任务的一部分。

一个任务，对于使用者是一个应用程序；对开发者则是贯穿活动着的任务的一个或多个视图，或者是一个活动栈。当设置 Intent.FLAG_ACTIVITY_NEW_TASK 标识启动一个活动意图时，任务就被创建了，这个意图被用作任务的根用途，定义区分哪个任务。如果活动启动时没有这个标识将被运行在同一个任务里（除非该活动以特殊模式被启动）。如果使用 FLAG_ACTIVITY_NEW_TASK 标识并且这个意图的任务已经启动，任务将被切换到前台而不是重新加载。

FLAG_ACTIVITY_NEW_TASK 必须小心使用。在用户看来，一个新的应用程序由此启动，但如果该应用程序不是用户所期望的，而且想要创建一个新的任务时，并且如果用户需要从桌面退出到他原来的地方然后使用同样的意图打开一个新的任务时，则需要使用新的任务标识。否则，如果用户在刚启动的任务里按桌面（Home）键而不是退出（Back）键，则用户的任务以及任务的活动将被放在桌面程序的后面，没有办法再切换过去。

3. 任务亲和力（task Affinity）

有些情况下 Android 需要知道某个任务的活动附属于哪个特殊的任务，即使该任务还没有被启动。这通过任务亲和力来完成，它为任务中一个或多个可能要运行的活动提供一个独一无二的静态名字，默认为活动命名的任务亲和力的名字就是实现该活动.apk 包的名字。这就提供了一种通用的特性，对用户来说所有在.apk 包里的活动都是单一应用的一部分。

当不带 Intent.FLAG_ACTIVITY_NEW_TASK 标识启动一个新的活动时，taskAffinity（任务亲和力）对新启动的活动没有影响，它将一直运行在它启动的那个任务里。但如果使用 NEW_TASK 标识，系统会检测已经存在的任务是否具有相同的亲和力。如果是，该任务会被切换到前台，新的活动会在任务的最上面被启动。

用户可以在其表现文件中的应用程序标签里为.apk 包里所有的活动设置自己的任务

亲和力,当然也可以为单独的活动设置标签。方法是:

(1) 如果.apk 包里包含多个用户可启动的上层应用程序,想要为每个活动分配不同的亲和力,则可以将不同的名字加上冒号附加在.apk 包名字的后面。例如,com.android.contacts 的亲和力命名可以是 com.android.contacts:Dialer 或 com.android.contacts:ContactsList。

(2) 如果想替换一个通知、快捷键或其他能从外部启动的应用程序的内部活动,则需要在想替换的活动里明确地设置 taskAffinity 属性。例如,如果想替换联系人详细信息浏览界面(用户可以直接操作或者通过快捷方式调用),则需要设置 taskAffinity 属性为 com.android.contacts。

4. 启动模式以及启动标识

控制活动和任务通信的最主要方法是设置启动模式的属性和意图相应的标识。这两个参数能以不同的组合来共同控制活动的启动结果,这在相应的文档里有描述,这里只介绍一些通用的用法以及几种不同的组合方式。

1) singleTop 模式

最通常使用的模式是 singleTop(除了默认为 standard 模式)。这种模式不会对任务产生任何影响,仅仅是防止在栈顶多次启动同一个活动。

2) singleTask 模式

singleTask 模式对任务的影响在于它能使活动总是在新的任务里被打开(或者将已经打开的任务切换到前台)。使用这个模式需要注意进程是如何和系统其他部分交互的,它可能会影响所有的活动。这个模式最好用于应用程序入口活动的标识中(支持 MAIN 活动和 LAUNCHER 分类)。

3) singleInstance 模式

singleInstance 启动模式更加特殊,该模式只能当整个应用只有一个活动时使用。

4) 启动标识

用户会经常遇到一种情况:其他实体如搜索管理器(SearchManager)或者通知管理器(NotificationManager)会启动其活动。这种情况下需要使用 Intent.FLAG_ACTIVITY_NEW_TASK 标识,因为活动在任务(还没有被启动)之外被启动。就像之前描述的那样,这时当前与任务和新的活动的亲和力匹配的任务将会切换到前台,然后在最顶端启动一个新的活动。当然也可以实现其他类型的特性。

5) 常用方法

一个常用的做法是将 Intent.FLAG_ACTIVITY_CLEAR_TOP 和 NEW_TASK 标识一起使用。这样如果用户的任务已经处于运行中,任务将会被切换到前台来,在栈里的所有活动除了根活动都将被清空,根活动的 onNewIntent(Intent) 方法传入意图参数后被调用。使用这种方法时经常 singleTop 或 singleTask 启动模式,这样当前实例会被置入一个新的意图,而不是销毁原先的任务然后启动一个新的实例。

另外可以使用的一个方法是设置活动的 taskAffinity 为空字串(表示没有亲和力),然后设置 finishOnBackground 属性。如果想让用户提供一个单独的活动描述通知,比较实用的做法是返回到应用的任务里。要指定这个属性,不管用户按 BACK 键还是 HOME 键,活动都会结束;如果这个属性没有指定,按 Home 键将会导致活动以及任务还留在系统里,并

且没有办法返回到该任务里。

5. 进程

在 Android 中，进程是应用程序的完整实现。其主要用途是：

(1) 提高稳定性和安全性，将不信任或者不稳定的代码移动到其他进程。

(2) 可将多个.apk 包运行在同一个进程里以减少系统开销。

(3) 帮助系统管理资源，将重要的代码放在一个单独的进程里，这样就可以单独销毁应用程序的其他部分。

进程的属性用来控制那些有特殊应用的组件运行的进程。注意这个属性不能违反系统安全，如果两个.apk 包不能共享同一个用户 ID 却试图运行在同一个进程里，这种情况是不被允许的，事实上系统将会创建两个不同的进程。

6. 线程

每个进程包含一个或多个线程。多数情况下，Android 避免在进程里创建多余的线程(除非创建其自己的线程)，保持了应用程序的单线程性，所有呼叫实例、广播接收器以及服务实例都是由这个进程里运行的主线程创建的。

注意新的线程不是为活动、广播接收器、服务或内容提供器实例创建的，这些应用程序的组件在进程里被实例化(除非另有说明，都在同一个进程处理)，实际上是进程的主线程。这说明当系统调用这些组件(包括服务)时不需要进程远距离或者封锁操作(就像网络呼叫或者计算循环)，因为这将阻止进程中的所有其他组件。用户可以使用标准的线程类或者 Android 的 HandlerThread 类去对其他线程执行远程操作。

以下是关于创建线程规则的例外：

(1) 呼叫 IBinder 或者 IBinder 实现的接口，如果该呼叫来自其他进程，则可以通过线程发送的 IBinder 或者本地进程中的线程池呼叫它们，从进程的主线程呼叫是不可以的。特殊情况下，呼叫一个服务的 IBinder 可以这样处理(虽然在服务里呼叫方法在主线程里已经完成)。这意味着 IBinder 接口的实现必须要有一种线程安全的方法，这样任意线程才能同时访问它。

(2) 呼叫由正在被调用的线程或者主线程以及 IBinder 派发的内容提供器指定的主方法，被指定的方法在内容提供器的类里有记录。这意味着实现这些方法必须要有一种线程安全的模式，这样任意其他线程可以同时访问它。

(3) 呼叫视图以及由视图里正在运行的线程组成的子类。通常情况下，这会被作为进程的主线程，如果创建一个线程并显示一个窗口，那么继承的窗口视图将从那个线程里启动。

2.2.3 创建一个 Hello Android 项目

创建一个 Android 新项目是很简单的，只要安装了 Android SDK，并且 Eclipse 软件版本在 3.2 或 3.3，那么就可以开始创建 New Android Project 了。

1. 创建一个新的 Android 项目

启动 Eclipse，执行 File→New→Project 菜单命令，如果已经安装好了 Android 的 Eclipse 插件，则会在弹出的 New Project 对话框中看到 Android Project 选项，如图 2-1 所示。

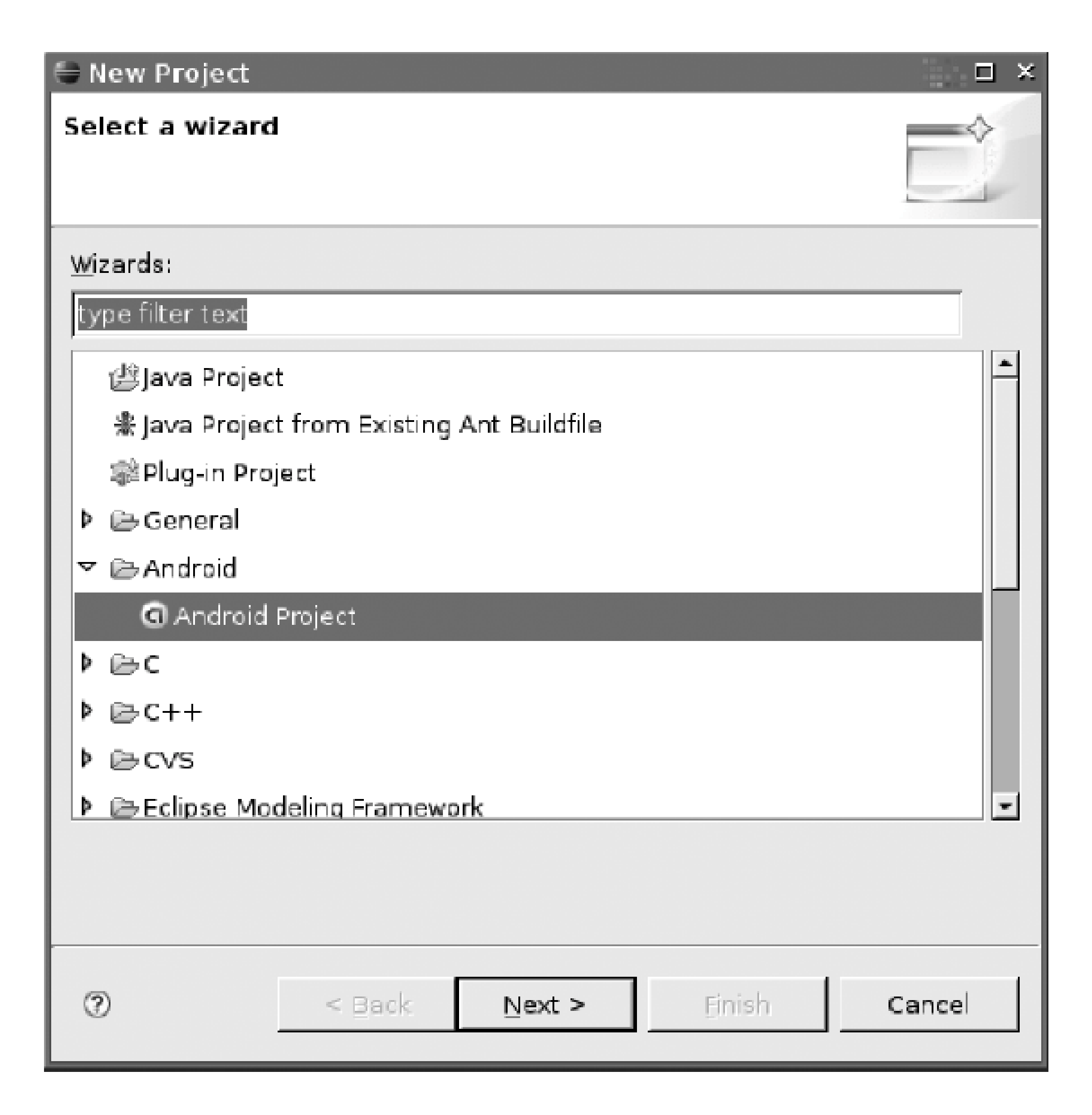


图 2-1　创建一个新的 Android 项目

选择 Android Project 选项，单击 Next 按钮。

2. 设置项目的细节参数

在 New Android Project 对话框中设置与项目有关的参数，如图 2-2 所示。其中：

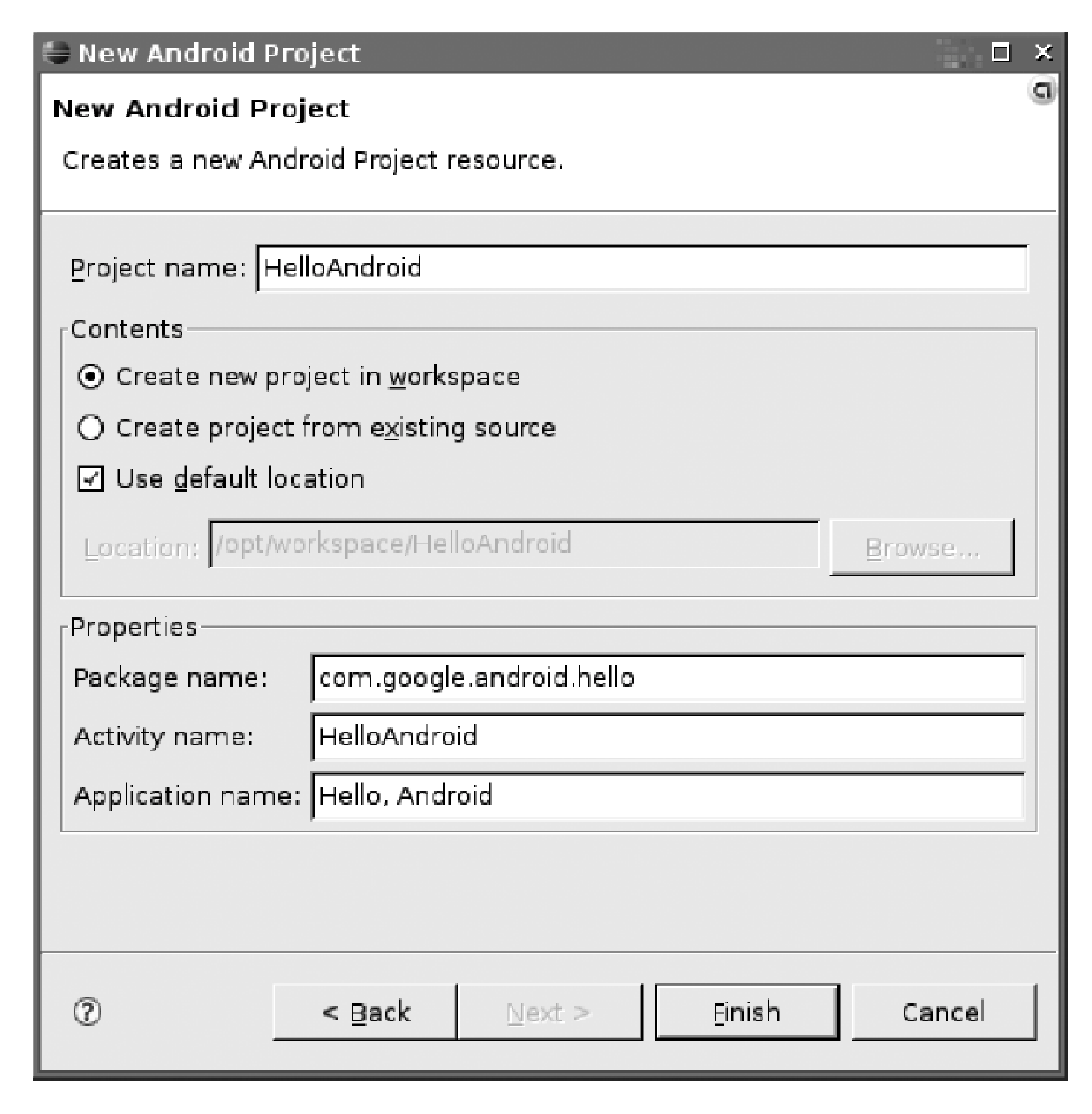


图 2-2　设置 Android 项目的细节参数

(1) Project Name 文本框中输入的是包含这个项目的文件夹的名称 HelloAndroid。

(2) Package Name 文本框中输入的是包名。遵循 Java 规范,用包名来区分不同的类是很重要的。图 2-2 中所示的包名是 com. google. android. hello。

(3) Activity Name 文本框中输入的是项目的主类名 HelloAndroid,这个类将会是 Android 的 Activity 类的子类。一个 Activity 类是一个简单的启动程序和控制程序的类,可以根据需要创建界面,但不是必须的。

(4) Application Name 文本框中输入的是在所创建应用程序上的一个易读标题"Hello,Android"。

(5) 选中 Contents 选项区域的 Create New Project in workspace 单选按钮,创建一个新的 Project。若选中 Use default location 复选框,则允许选择一个已存在的项目。这里创建一个新的 Android Project。

3. 编辑自动生成的代码

1) 自动生成的代码

当项目创建后,即刚才创建的 HelloAndroid 就会包含如下代码:

```
public class HelloAndroid extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle)
    {
        super.onCreate(icicle);
        setContentView(R.layout.main);
    }
}
```

2) 修改代码

当一个项目建立好以后,最直接的效果就是在屏幕上显示一些文本,下面是修改完成后的代码,稍后再逐行解释。

```
import android.widget.TextView
public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

(1) 文本标签 TextView:在 Android 程序中,用户界面是由 Views 类来组织的。一个 View 可以简单地理解为可以绘制的对象,如选择按钮、一个动画或者一个文本标签(本程序中),这个显示文本标签的 View 子类叫做 TextView。

（2）构造一个 TextView：

```
TexiView tv = new TextView(this);
```

TextView 的构造参数是 Android 程序的 Context 实例。Context 可以控制系统调用，它提供了诸如资源解析、访问数据库等。Activity 类继承自 Context 类，因为 HelloAndroid 是 Activity 的子类，所以它也是一个 Context 类，因此这里能用 TextView(this)。

（3）当构造完 TextView 后需要告诉它显示什么：

```
tv.setText("Hello, Android");
```

（4）最后要把 TextView 显示在屏幕上：

```
setContentView(tv);
```

Activity 的 setContentView()方法指示出系统要用哪个 View 作为 Activity 的界面。如果一个 Activity 类没有执行这个方法，将会没有界面并且显示白屏。由于在本程序中要显示文本，所以传入已创建好的 TextView。

3）运行 HelloAndroid 程序

使用 Android 的 Eclipse 插件就可以很轻松地运行刚刚创建完成的程序。

（1）执行 Run→Open Run Dialog 菜单命令，打开 Run 对话框，如图 2-3 所示。

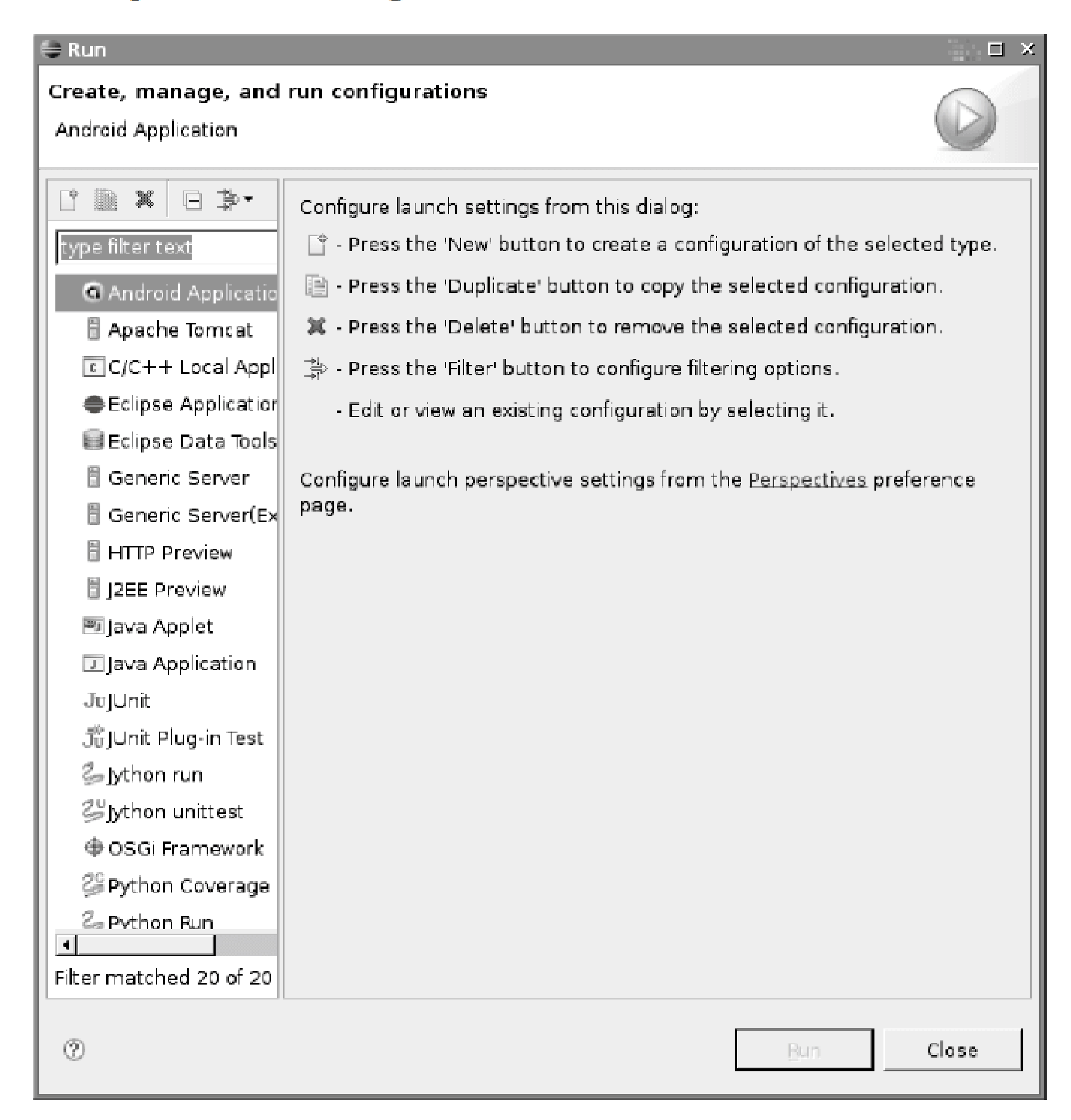


图 2-3　Run 对话框

(2) 高亮 Android Application 标签，然后单击左上角的图标；或者直接双击 Android Application 标签，在 Run 对话框中的右侧区域将会看到一个新的运行项目，Name 为 New_configuration，如图 2-4 所示。

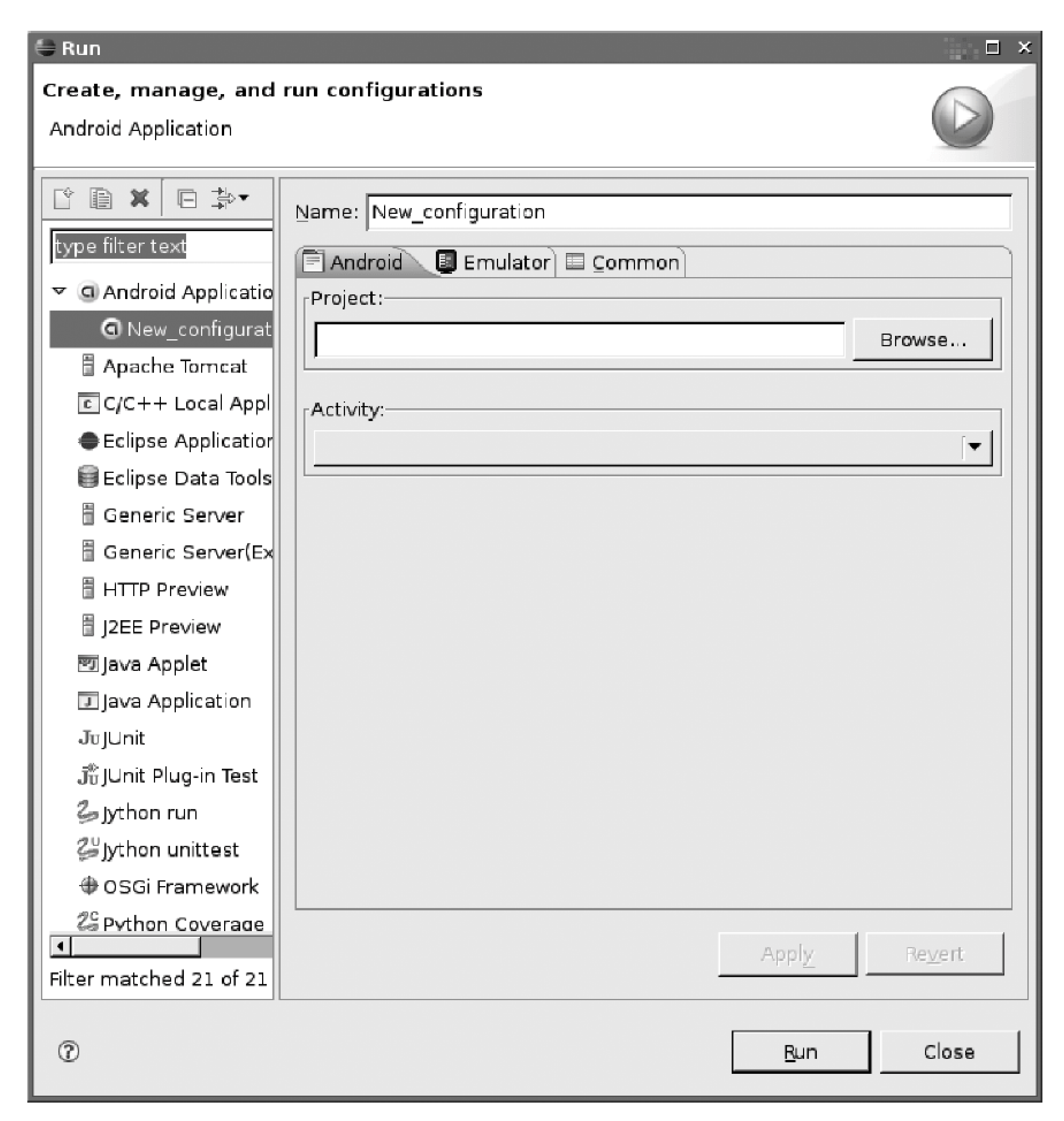


图 2-4 新的运行项目

(3) 在 Name 文本框中输入一个可以表意的名称，例如“Hello, Android”。

(4) 单击 Project 区域的 Browse 按钮选取创建好的项目(如果在 Eclipse 中有很多个项目，确保选择要运行的项目)，插件会自动搜索在项目中的 Activity 类并且将所有找到的添加在 Activity 区域的下拉列表框中。这里只有 HelloAndroid 一个项目，所以会将其作为默认选择，如图 2-5 所示。

(5) 单击 Apply 按钮，如图 2-5 所示。

(6) 单击 Run 按钮，Android Emulator 将会启动，HelloAndroid 应用程序的易读标题(也是源代码)“Hello，Android”就会被显示出来，如图 2-6 所示。

至此完成了 HelloAndroid 程序的创建及运行；下面介绍如何把界面的布局用 XML 表示以及使用命令行进行编译和运行 Android 程序。

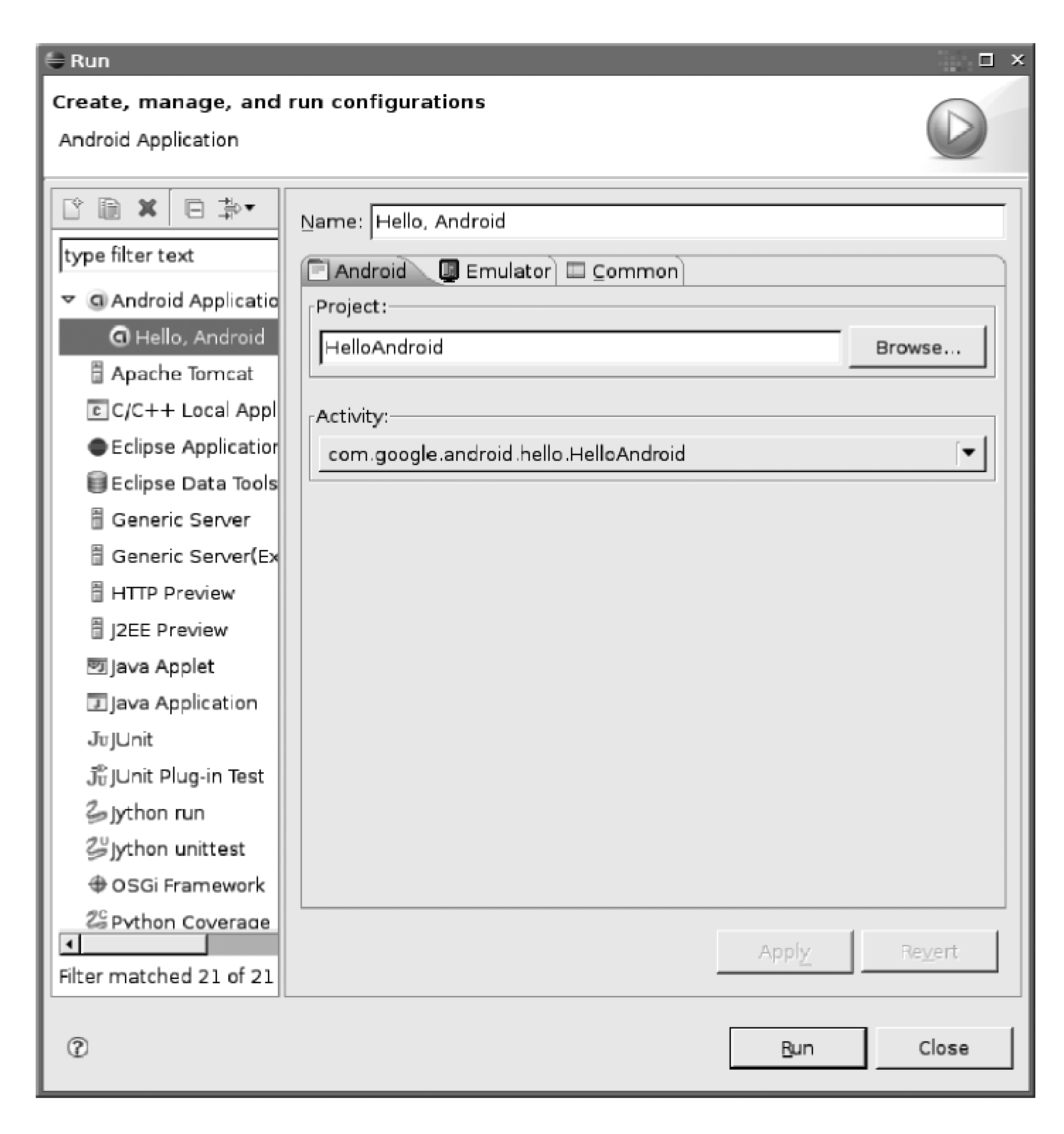


图 2-5　设置运行项目 HelloAndroid



图 2-6　启动 Android Emulator 运行 HelloAndroid 程序

2.2.4 将界面实现用 XML 编排

刚刚完成的 HelloAndroid 程序例子称为程序化的界面编排，意思是说构建的应用程序界面是直接使用的源代码。读者完成很多界面程序后就会熟知此类方式的脆弱性：一个对布局的小小修改就可能导致代码出错并浪费调试时间。

为了解决这个问题，Android 提供了一种可替换的界面构建方式：基于 XML 的布局文件。

1. Android XML 布局文件

对刚才创建的 HelloAndroid 项目用 XML 修改自动生成的代码，然后再进行演示，可达到相同的界面效果。代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< TextView xmlns:android = "http://schemas.android.com/apk/res/android"
android:layout_width = "fill_parent"
android:layout_height = "fill_parent"
android:text = "Hello, Android"/>
```

2. Android XML 布局文件的结构

Android XML 布局文件的大体结构很简单，它是一个标签的树，任何一个标签就是 View 类的名字。在这个例子中，它是一个很简单的只有一个元素的树，一个 TextView。可以使用任何继承自 View 类的名字作为标签的名字，包括在代码中自定义的 View 类。

这种结构可以很容易地构建界面，比在源代码中使用的结构和语法更简单。它实际上是可以将界面和应用程序逻辑分离的模式，这种模式的设计灵感来自于 Web 开发。

3. 布局文件中 XML 的属性

在 XML 布局文件中，XML 的属性含义如下：

(1) xmlns:android 是 XML 命名空间的声明，用于告诉 Android 的工具其将要涉及的公共属性已被定义在 XML 命名空间。在每一个 Android XML 布局文件的最外边的标签必须有这个属性。

(2) android:layout_width 属性定义了在屏幕上这个 View 可用的宽度是多少。

(3) android:layout_height 属性定义了在屏幕上这个 View 可用的高度是多少。

(4) android:text 设置 TextView 所包含的文本内容，当前的文本信息为"Hello, Android"。

4. XML 布局文件的存放位置

XML 布局文件需要放在项目目录的 res/文件夹下。res 是 resources 的缩写，是存放所有非代码资源的文件夹，包含图片、本地化字符串和 XML 布局文件。

这些 Eclipse 的插件已经创建好了，在上面的例子中并没有使用。使用方法是在包浏览器中展开 res/layout，并且编辑 main.xml 文件，替换掉原来的文本内容，然后保存。

5. 使用 Eclipse 的插件生成 XML 布局文件

(1) 在包浏览状态打开代码文件夹中的 R.java 文件，将会看到如下内容：

```
public final class R {
  public static final class attr {
  };
  public static final class drawable {
    public static final int icon = 0x7f020000;
  };
  public static final class layout {
    public static final int main = 0x7f030000;
  };
  public static final class string {
    public static final int app_name = 0x7f040000;
  };
};
```

(2) 一个项目的 R.java 文件是一个定义所有资源的索引文件，使用这个类就像使用一种速记方式来引用项目中包含的资源，可以快速地、互动式地定位正在寻找的特定引用。

(3) 现在需要注意的是叫做 layout 的内部类和其成员变量 main。插件会通知用户添加一个新的 XML 布局文件，然后重新产生这个 R.java 文件。例如用户添加了新的资源到项目中，这时会看到 R.java 也相应地改变了。

(4) 修改 HelloAndroid 源代码，使用新的 XML 布局界面。替换掉编码式的界面模式。下面所示为新代码，可以看到代码变得更加简单了。

```
public class HelloAndroid extends Activity {
public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
}
}
```

需要说明的是，当做这些改变时，不要仅仅复制、粘贴代码，多体验 R.java 的代码编译特点，这对设计编程有很大的帮助。

(5) 完成以上修改后重新运行 HelloAndroid 程序，发现两种不同的界面编排方式会产生同样的效果。

2.2.5 调试项目

在 HelloAndroid 源代码中制造一个 bug，这时的代码如下：

```
public class HelloAndroid extends Activity {
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);
  Object o = null;
  o.toString();
  setContentView(R.layout.main);
}
}
```

这个简单的变化会引起一个 NullPointerException 异常，如果再次运行程序则会看到如图 2-7 所示的屏幕信息。



图 2-7　调试时产生的异常

要找到是什么地方出错，需要在源代码的"Object o = null;"行后(可以双击在 Eclipse 中显示行数的左部区域)设置一个断点，然后执行 Run→Debug 菜单命令并选择最后一次的运行加载。程序将会重启 Emulator，但是这个时候它会挂起，当执行到断点位置时，在 Eclipse 的调试模式视图中停止在该行代码处，如图 2-8 所示。

HelloAndroid.java - HelloAndroid/src/com/google/android

```
package com.google.android.hello;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        Object o = null;
        o.toString();
        setContentView(R.layout.main);
    }
}
```

HelloAndroid/src/com/google/android/hello/HelloAndr

图 2-8　Eclipse 调试模式视图

2.3 Android 开发者联盟

2.3.1 开发基于 Android 平台的应用

在 Android 平台上可以开发出各种各样的应用。Android 的应用程序是用 Java 语言开发的,因此只要会 Java 语言的学习就可以很快地进入 Android 开发领域。同时 Android 平台提供了 2D、3D 的图形支持、数据库支持(SQLite),并且集成了浏览器。基于 Android 这个开放的平台可以开发出丰富多彩的应用,包括工具、管理、Internet 和游戏等,这些都取决于程序员的自由发挥和创意。

2.3.2 参加 Android 开发者大赛

Google 公司为了吸引更多的开发者参与到 Android 的开发中,举办了每次奖金为 1000 万美元的开发者竞赛,如果能开发出创意十足、非常有用的软件,那么在今后的 Google 及各公司的开发者大赛中都可以大显身手。随着 Andriod 系统的发展,Andriod 开发设计大赛正如火如荼地展开。

为了促进产业的发展,拓展应用市场,一些大企业推出了一系列的 Andriod 开发设计大赛,这些企业不只是手机终端应用开发公司,专注于电视产品的公司对此也有巨大兴趣。Google 作为 Andriod 系统的研发公司,他们主办的 Andriod 设计大赛自然是最有影响力和质量的。与此同时各大高校为促进学生就业能力的提高,也举办了生动活泼的 Andriod 开发者设计大赛。表 2-2 所示就是一个基于 Android 的移动应用开发设计案例。

表 2-2 基于 Android 的移动应用开发设计案例

基于 Android 的移动应用开发	
赛题简介:介绍整个赛题的思路和整体要求	开发一个基于 Android 平台的手机相册软件。该软件可以拍摄、编辑、查看、分享本地相册以及网络相册
赛题业务场景:描述赛题相关的真实企业业务背景,从真实场景中适当简化或者提炼出适合比赛的赛题场景	业务模型可以参考人人网的网络相册,提供在线浏览、下载、编辑、上传照片等功能,也可即时分享照片到另外一个 SNS 社区(新浪微博、腾讯微博等)
功能性需求	(1) 具有拍摄、编辑、查看、共享及分发的功能。 (2) 连接 SNS 网络时使用客户端模式,禁止使用 Browser 或者 WebView 等借助网页页面的方式。 (3) 分享 SNS 社区可以从新浪微博、网易微博、腾讯微博中选择一个或多个,也可以在此范围之外选择。 (4) 能够绑定 SNS 社区账户,第一次访问需要授权及认证,以后便可直接访问。 (5) 具有换肤功能,提供换肤模板库供用户选择。 (6) 网络异常时能够提示或禁止相关网络相册功能。 (7) 自动切换 3G、Wi-Fi 等网络通道,优先使用 Wi-Fi

续表

非功能性需求	(1) 实际并发用户数不低于 50 个。 (2) 平均事务响应时间小于等于 10 秒。 (3) 平均服务响应时间小于等于 5 秒。 (4) 文件传输不大于 4MB。 (5) 虚拟并发用户数为 100～300。
其他限制条件：开发环境、实验平台、开发语言、数据库、编译器等限制条件	开发环境：Android SDK 2.2、ADT、Windows Mobile 5.0。 开发平台：ECLIPS 或 Visual Studio 2010 以上。 开发语言：J2ME/NET/Compact Framework/HTML+JS/PHP。 数据库：mySQL/SQLite
测试数据或平台：提供给参赛者的测试环境和测试数据(可提供电子档)	测试平台：ADT(Android Development Tools)、Activity、Intent、Service、ContentProvicer。 测试数据：可在人人网上模拟操作，记录测试数据或从 ContentProvider 中获取
其他要求	文档要求：概要设计说明书(描述软件系统结构、逻辑结构、物理结构、部署结构、功能结构及关键技术，关键业务模块需通过 UML 图(用例图、时序图、状态图、包图、主要类图等)进行详细描述)、需求规格说明书(包括功能设计、非功能性设计、系统用例)。 测试要求：需进行单元测试，提供单元测试用例，单元测试覆盖率不低于 90%；提供性能测试文档(包括测试脚本、实际吞吐率、阈值等)

2.3.3 Android 得到更多人的认可和尊重

在这个开放的平台上，任何程序员开发的应用程序都可以放在 Android Market 上，供所有的 Android 手机用户下载体验，用户都可以对此应用加以评价和使用。这是一个展示个人能力和技术魅力的平台。软件开发的个人英雄主义时代将在 Android 移动平台上完美地体现出来，程序员完全可以凭自己的能力开发出一款得到更多赞誉的应用。

2.3.4 Android Market

Android Market 是一个由 Google 公司为 Android 系统用户创建的服务，允许安装了 Android 系统的手机和平板电脑用户从 Android Market 浏览和下载一些应用程序。用户可以购买或免费试用这些应用程序。

1. 新版 Android Market

2012 年 3 月 7 日 Google 把在线商店 Android Market 更名为 Google Play Store，于 13 日起正式启用。老版本的 Google Android Market 应用店在组织和搜索上的体验并不好，这种状况终于得到了改善，新的应用商店首次允许用户通过浏览器搜索和购买应用软件，更容易搜索和浏览，并贴有图片和用户评价，设计上更精美。此前的 Android 手机用户只能通过手机上的客户端访问 Android 电子市场，网页端的功能非常弱，新版本的 Android Market 具有更好的界面和功能，有方便的目录导航、最热门应用的排序，支持 Android 用户登录并允许用户直接将应用下载至手机，用户可以在网页上查看自己以往的安装下载记录。

2. 开发者发布应用程序至 Android Market

开发者感兴趣的应用开发：社交类的；媒体消费、管理、修改或共享；效率与协助类的，

如电子邮件、即时通信、日历等；游戏、资讯与信息；突破传统用户界面的新构想、混搭式网络服务 Mash-up、定位服务、社会公益、服务于全球经济发展的各类应用等；以及任何开发者所热衷的领域。

要发布 Android 应用程序至 Android Market，必须先支付一笔注册费。以下步骤说明如何注册 Android Market Account、sign（签署）应用程序并发布应用程序至 Market 的过程。

（1）注册 Google 账户并登录。

（2）用信用卡支付 25 美元注册费用。

（3）导出未签署的 APK 程序。

（4）建立 keystore。

（5）执行 Export Signed Spplication Package。

（6）浏览专案名称并确认。

（7）单击 Browse 并选择 keystore。

（8）利用下拉选项选择可用的 Alias，并输入方才建立 keystore 所使用的密码。

（9）建立 signed apk 完成。

（10）正确上传 Market 之后所显示的程序 ICON 图标。

（11）输入应用程序简介与 Title。

（12）正确上传应用程序至 Market。

只要程序员开发的应用软件足够吸引人，让用户有足够的理由购买，那么就可以获得很好的回报，从而促进更好的应用开发。

第3章　深入认识 Android 系统

本章深入介绍 Android 系统，加深对 Android 系统的认识。通过本章学习，读者应该掌握以下内容：

(1) Android 系统结构和初始化流程。

(2) Android 系统的 Linux 内核和驱动程序。

(3) Android 内核深度解析。

(4) Android 底层库和程序。

(5) Android 的进程通信机制。

3.1　Android 系统结构和初始化过程

3.1.1　Android 系统结构

Android 是为移动设备设计的软件平台，包括操作系统、中间件和一些关键应用。Android SDK 提供了必需的工具和进行应用开发所必需的 Java API。Android 是一个开放的软件系统，为用户提供了丰富的移动设备开发功能，其系统结构从下至上包括 4 个层次，分别为 Linux 内核层、系统运行库层、应用程序框架层和应用程序层，如图 3-1 所示。

1. Linux 内核层

第一层是 Linux 内核层，包括 Linux 操作系统及驱动，依赖于 Linux 2.6 内核，如安全性、内存管理、进程管理、网络协议栈和驱动模型。不支持 Linux 2.4 内核。例如 Android 1.0(Release-1.0)使用 Linux 2.6.25，Android 1.6(SDK-1.6)使用 Linux 2.6.29。除了标准的 Linux 内核外，Android 系统还增加了 Binder IPC 驱动、Wi-Fi 驱动、蓝牙驱动等驱动程序，Linux 内核也同时作为硬件和软件栈之间的抽象层。

1) Java

Java 是一种简单的、面向对象的、分布式的、解释的、健壮的、安全的、可移植的语言，在众多领域得到了广泛应用，尤其是在分布式的、大型的网络领域。使用 Java 语言以及相关的 Java 技术，可以快速地构建出大型的、平台独立的系统。

但 Java 的可移植性特点，导致了 Java 不能够直接与计算机硬件交互的缺陷。但是，通过不同操作系统的 Java 虚拟机(JVM)，Java 可以间接地实现与计算机基本硬件交互的目的。而不同平台的 JVM 内核必须通过其他语言来实现底层硬件的驱动和交互。所以，如果 Java 开发的应用要驱动硬件或与硬件交互，就必须依赖其他可以驱动硬件或与硬件交互的语言。

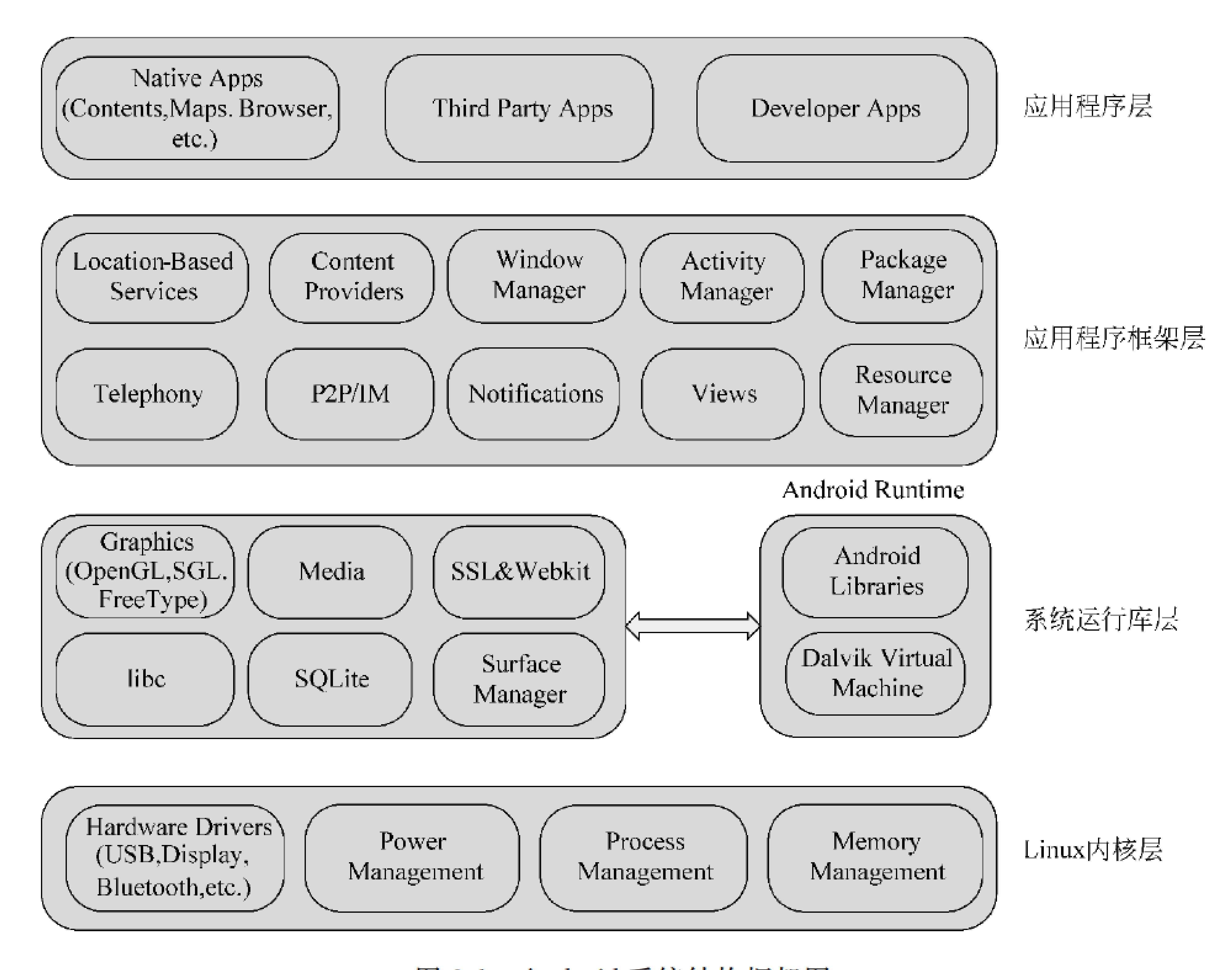


图 3-1　Android 系统结构框架图

2）C 语言

C 语言是驱动硬件、与硬件交互方面的首选。由于 C 语言可以像汇编语言一样对位、字节和地址进行操作，因而可以直接与硬件进行交互。C 语言把高级语言的基本结构和语句与低级语言的实用性紧密结合，使得 C 语言成为使用最简单、功能最强大的中级语言。当前几乎所有的操作系统的内核硬件驱动与交互都是用 C 语言实现的。

3）Java 与 C 语言之间的数据交换

在用 Java 开发与硬件有关的应用时，必须面对和处理 Java 与硬件驱动语言之间的数据交换，而 Java 与 C 语言之间的数据交换则是这方面应用最普遍、通用的实现和示例。

Java 与 C 语言之间的数据交换可以归纳为 Java 本地化方法、文件方法、网络方法、命名管道方法 4 种。这里采用 Java 本地化方法，具体过程如图 3-2 所示。

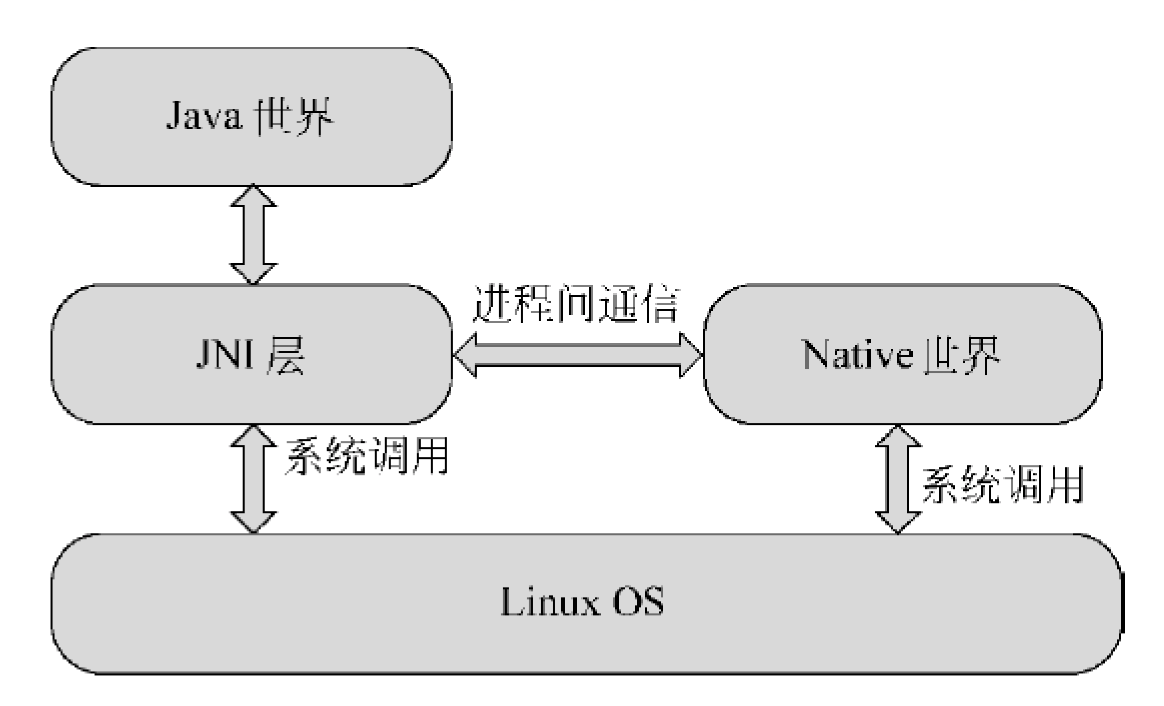


图 3-2　Java 世界与 Native 世界的交互

Java 本地化(Java Native Interface,JNI)是 JDK 的一部分,用来给 Java 世界提供一个本地代码的接口。通过 JNI,一方面可以使运行在 JVM 上的 Java 代码操纵其他语言(如 C/C++)编写的程序库;另一方面可以把 JVM 嵌入到本地代码中,并在本地代码(如 C/C++)中调用 Java。

通过 Java 本地化,Java 和 C 语言之间可以通过函数之间的参数进行数据交换,这也是 Java 和 C 语言之间最基本的数据交换方式。大部分平台下的 JVM 就是通过 C 语言实现对计算机硬件的驱动和交互,并提供 Java 访问的接口和方法,从而实现 Java 代码对基本硬件的交互,如鼠标、键盘等事件的响应。

实际应用中,本地化方法也是 Java 应用系统驱动硬件的基本方法。如用 C 语言实现对具体硬件的驱动,而 Java 应用系统通过本地化方法把与硬件交互所需的数据以参数的方式传递给驱动硬件的 C 语言函数,驱动硬件的 C 语言函数把结果再以函数返回值的方式传递给 Java 应用系统,从而完成与硬件交互过程中 Java 与 C 语言之间的数据交换。

2. 系统运行库层

第二层是系统运行库层,主要包括两个部分程序库(Libraries)和 Android 运行时库(Android Runtime)。它们可以通过 Java 本地调用 JNI(Java Native Interface)的接口函数实现和上层之间的通信。

(1) 程序库(Libraries):Android 包含一些 C/C++库,这些库能够被 Android 系统中的不同组件使用,它们通过应用程序框架来为开发者提供。

(2) Android 运行时库(Android Runtime):分核心库和 Dalvik 虚拟机两个部分。核心库提供了 Java 语言核心库的大多数功能,主要通过 JNI 的方式向应用程序框架层提供调用底层程序库的接口。Dalvik 虚拟机是为了能同时高效地运行多个 VMs(Virtual Machines)而实现的,它是基于寄存器的,所有的类都经由 Java 汇编器编译,然后通过 SDK 中的 dx 工具转换成 .dex 格式并由虚拟机执行。

3. 应用程序框架层

第三层是应用程序框架层,包含所有开发所用的 SDK 类库和某些未公开接口类库的框架层,是整个 Android 平台核心机制的体现。开发人员可以利用这套应用框架开发出很好的应用程序。应用框架的主要部分如下。

(1) Views(UI 组件):可扩展显示,可构建 UI。

(2) Content Providers:用于在各应用之间共享数据。

(3) Resource Manager:管理非代码资源。

(4) Notification Manager:显示用户提示和状态栏。

(5) Activiy Manager:管理运行应用程序。

4. 应用程序层

第四层是应用程序层。Android 的应用程序通常涉及用户界面和用户交互,这类程序是用户实实在在能感觉得到的。Android 的应用程序目前以 Java 进行编写。

Android 本身提供了桌面、联系人、电话、浏览器等众多核心应用。开发者还可以使用应用程序框架层的 API 实现自己的程序。系统部分应用和第三方开发的应用都是位于这个层次上,但两者不完全相同,其中系统应用会用一些隐藏的类,而第三方的应用是基于 SDK 基础开发的。一般 Android 开发是在 SDK 基础上用 Java 编写应用程序,但本机开发

程序包 NDK 提供了应用层穿越 Java 框架层直接和底层包含了 JNI 接口的 C/C++库直接通信的方法。

5. 各层的实现及关联

第一层由 C 语言实现，第二层由 C 语言和 C++语言实现，第三、四层主要由 Java 代码实现。

从 Linux 操作系统的角度来看，第一、二层之间是内核空间与用户空间的分界线，第一层运行于内核空间，第二、三、四层运行于用户空间；第二、三层之间是本地代码层和 Java 代码层的接口；第三、四层之间是系统 API 接口。

3.1.2 Android 系统的初始化过程

为了更加直观地介绍 Android 的初始化过程，引用 Android 系统的初始化流程图，如图 3-3 所示。

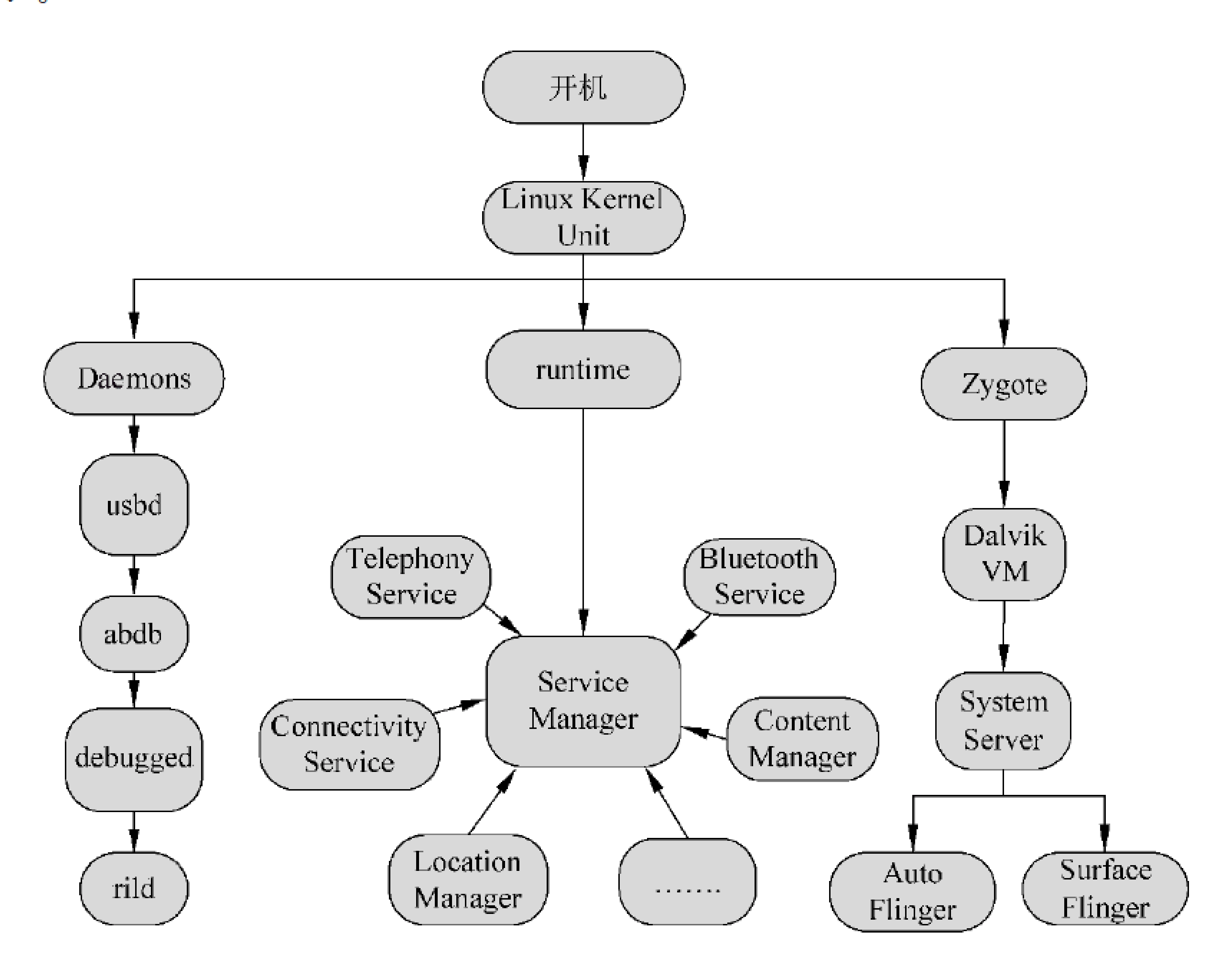


图 3-3 Android 系统的初始化流程图

1. Android 系统的初始化(启动)过程

(1) 首先会启动 Linux 基础系统，然后引导加载 Linux Kernel 并启动初始化进程(Init)。

(2) 接着启动 Linux 守护进程(Daemons)。

(3) 在启动 Linux 守护进程的同时还需要启动 Zygote 进程。

(4) 再接着，需要初始化 runtime 进程。

(5) runtime 进程初始化后，将发送一个请求到 Zygote，开始启动系统服务。这时 Zygote 将为系统服务进程建立一个虚拟机实例，并启动系统服务。

(6) 系统服务将启动原生系统服务，主要包括 Surface Flinger 和 Audio Flinger。这些服务将注册到服务管理器(Service Manager)作为 IPC 服务目标。

(7) 系统服务将启动 Android 管理服务,Android 管理服务都将被注册到服务管理器上。

(8) 当系统加载完所有的服务之后会处于等待状态,但是每个应用程序都将启动一个单独的进程。这时,系统将启动一个 Home 进程和一个 Contacts 进程,各个进程通过 IPC 机制进行交互。

到这里,Android 系统的整个启动过程就结束了,可以在该系统中运行应用程序了。

2. 对图 3-3 中的几个进程做简单解释

(1) usbd: USB 守护进程,管理 USB 连接。

(2) adbd: Android Debug Bridge 守护进程,管理 ADB 连接。

(3) debugged: Debug 守护进程,管理调试进程请求(包括内存转换等)。

(4) rild: 无线接口守护进程,管理无线通信。

(5) Zygote: 初始化一个 Dalvik 虚拟机实例; 装载 Socket 请求所需的类和监听; 创建虚拟机实例来管理应用程序进程。

(6) runtime: 初始化服务管理器; 注册服务管理器,以其作为默认 Binder 服务的 Context 管理器。

(7) Service Manager: 服务管理器,所有的 Server(System Server)都需要向其注册,应用程序需要向其查询相应的服务。

(8) 其他是一些系统服务,这里就不一一介绍了。

实际上,Android 系统的启动过程就是从 Android 系统底层的 Linux 内核层一步步加载和注册到应用程序框架层,最终在应用程序层运行自己的 APP(应用软件)。

3.2 Android 系统的 Linux 内核和驱动程序

3.2.1 Android 系统的 Linux 内核

Linux 内核可以进一步划分为三层: 最上面是系统调用接口,实现一些基本的功能,例如 read 和 write 等; 系统调用接口之下是内核代码,可以更精确地定义为独立于体系结构的内核代码,这些代码是 Linux 所支持的所有处理器体系结构所通用的; 在这些代码之下是依赖于体系结构的代码,构成了通常称为 BSP(Board Support Package,板级支持包)的部分,这些代码用作给定体系结构的处理器和特定于平台的代码。

1. Linux 内核的属性

在讨论大型而复杂系统的体系结构时,可以从很多角度来审视该系统,体系结构分析提供了一种更好地理解源代码的方法。

Linux 内核实现了很多重要的体系结构属性。在或高或低的层次上,内核被划分为多个子系统。Linux 也可以看作一个整体,因为它会将所有这些基本服务都集成到内核中。这与微内核的体系结构不同,后者会提供一些基本的服务,例如通信、I/O、内存管理和进程管理,更具体的服务都是插入到微内核层中的。

Linux 内核在内存和 CPU 使用方面越来越具有较高的效率,非常稳定并且具有良好的可移植性。Linux 编译后可在大量处理器和具有不同体系结构约束与需求的平台上运行。

例如 Linux 可以在一个具有内存管理单元(MMU)的处理器上运行,也可以在那些不提供 MMU 的处理器上运行(Linux 内核的 uClinux 移植提供了对非 MMU 的支持)。

2. Linux 内核的主要子系统

1) 系统调用接口

SCI 层提供了执行从用户空间到内核的函数调用机制。这个接口依赖于体系结构,甚至在相同的处理器家族内也是如此。SCI 实际上是一个非常有用的函数调用多路复用和多路分解服务。在 ./linux/kernel 目录中可以找到 SCI 的实现,并在 ./linux/arch 目录中找到依赖于体系结构的部分。有关这个组件的更详细信息可以参考相关资料。

2) 进程管理

进程管理的重点是进程的执行。在内核中,这些进程称为线程,代表了单独的处理器虚拟化(线程代码、数据、堆栈和 CPU 寄存器);在用户空间,通常使用进程这个术语。不过 Linux 实现并没有区分进程和线程这两个概念。内核通过 SCI 提供了一个 API(应用程序编程接口)来创建一个新进程(fork、exec 或 POSIX(Portable Operating System Interface)函数),停止进程(kill、exit),并在它们之间进行通信和同步(signal 或 POSIX 机制)。

进程管理还包括处理活动进程之间共享 CPU 的需求。内核实现了一种新型的调度算法,不管有多少个线程在竞争 CPU,这种算法都可以在固定时间内进行操作,这种算法称为 O(1) 调度程序。这个名字就表示它调度多个线程所使用的时间和调度一个线程所使用的时间是相同的。O(1) 调度程序也可以支持多处理器(称为对称多处理器或 SMP)。可以在 ./linux/kernel 目录中找到进程管理的源代码,在 ./linux/arch 目录中找到依赖于体系结构的源代码。有关这个算法的更多内容可查阅相关参考资料。

3) 内存管理

内核所管理的另外一个重要资源是内存。为了提高效率,如果由硬件管理虚拟内存,内存是按照所谓的内存页方式进行管理的(对于大部分体系结构来说都是 4KB)。Linux 包括了管理可用内存的方式以及物理和虚拟映射所使用的硬件机制。

不过内存管理要管理的可不止 4KB 缓冲区。Linux 提供了对 4KB 缓冲区的抽象,例如 slab 分配器。这种内存管理模式使用 4KB 缓冲区为基数,然后从中分配结构,并跟踪内存页使用情况,例如哪些内存页是满的,哪些页面没有完全使用,哪些页面为空,这样就允许该模式根据系统需要来动态地调整内存使用。

为了支持多个用户使用内存,有时会出现可用内存被消耗光的情况。由于这个原因,页面可以移出内存并放入磁盘中,这个过程称为交换,因为页面会被从内存交换到硬盘上。内存管理的源代码可以在 ./linux/mm 目录中找到。

4) 虚拟文件系统

虚拟文件系统(VFS)是 Linux 内核中非常有用的一个方面,因为它为文件系统提供了一个通用的接口抽象。VFS 在 SCI 和内核所支持的文件系统之间提供了一个交换层。在 VFS 上面是对诸如 open、close、read 和 write 等函数的一个通用 API 抽象;在 VFS 下面是文件系统抽象,它定义了上层函数的实现方式;它们是给定文件系统(超过 50 个)的插件。文件系统的源代码可以在 ./linux/fs 目录中找到。

文件系统层之下是缓冲区缓存,它为文件系统层提供了一个通用函数集(与具体文件系统无关)。这个缓存层通过将数据保留一段时间(或者随即预先读取数据以便在需要时可

用)优化了对物理设备的访问。缓冲区缓存之下是设备驱动程序,它实现了特定物理设备的接口。

5) 网络堆栈

网络堆栈在设计上遵循模拟协议本身的分层体系结构。回想一下,Internet Protocol (IP) 是传输协议(通常称为传输控制协议或 TCP)下面的核心网络层协议。TCP 上面是 Socket 层,它是通过 SCI 进行调用的。

Socket 层是网络子系统的标准 API,它为各种网络协议提供了一个用户接口。从原始帧访问到 IP 协议数据单元(PDU),再到 TCP 和 User Datagram Protocol (UDP),Socket 层提供了一种标准化的方法来管理连接,并在各个终点之间移动数据。内核中网络源代码可以在 ./linux/net 目录中找到。

6) 设备驱动程序

Linux 内核中有大量代码都在设备驱动程序中,它们能够运转特定的硬件设备。Linux 源码树提供了一个驱动程序子目录,这个目录又进一步划分为各种支持设备,例如 Bluetooth、I2C、serial 等。设备驱动程序的代码可以在 ./linux/drivers 目录中找到。

7) 依赖体系结构的代码

尽管 Linux 很大程度上独立于所运行的体系结构,但是有些元素则必须考虑体系结构才能正常操作并实现更高效率。./linux/arch 子目录定义了内核源代码中依赖于体系结构的部分,其中包含了各种特定于体系结构的子目录(共同组成了 BSP)。对于一个典型的桌面系统来说,使用的是 i386 目录。每个体系结构子目录都包含了很多其他子目录,每个子目录都关注内核中的一个特定方面,例如引导、内核、内存管理等。这些依赖体系结构的代码可以在 ./linux/arch 目录中找到。

3. Linux 内核的其他有用特性

如果觉得 Linux 内核的可移植性和效率还不够好,Linux 还提供了其他一些特性,它们无法划分到上面的分类中。

作为一个生产操作系统和开源软件,Linux 是测试新协议及其增强的良好平台。Linux 支持大量网络协议,包括典型的 TCP/IP,以及高速网络的扩展(大于 1GB 和 10GB 以太网)。Linux 也可以支持诸如流控制传输协议(SCTP)等协议,它提供了很多比 TCP 更高级的特性(是传输层协议的接替者)。

Linux 还是一个动态内核,支持动态添加或删除软件组件(称为动态可加载内核模块)。它们可以在引导时根据需要(当前特定设备需要这个模块)或在任何时候由用户插入。

Linux 最新的一个增强是可以用作其他操作系统的操作系统(称为系统管理程序)。Linux 对内核进行了修改,称为基于内核的虚拟机(KVM)。这个修改为用户空间启用了一个新的接口,允许其他操作系统在启用了 KVM 的内核之上运行,除了运行 Linux 的其他实例之外,Microsoft Windows 也可以进行虚拟化,唯一的限制是底层处理器必须支持新的虚拟化指令。

3.2.2 Android 系统的驱动程序

操作系统的目的之一就是将系统硬件设备细节从用户视线中隐藏起来。例如虚拟文件系统对各种类型已安装的文件系统提供了统一的视图而屏蔽了具体底层细节。本节介绍

Linux 核心对系统中物理设备的管理。

1. 设备驱动

1）设备驱动的概念

CPU 并不是系统中唯一的智能设备，每个物理设备都拥有自己的控制器。键盘、鼠标和串行口由一个高级 I/O 芯片统一管理，IDE 控制器控制 IDE 硬盘，SCSI 控制器控制 SCSI 硬盘，等等。每个硬件控制器都有各自的控制和状态寄存器（CSR）并且各不相同，例如 Adaptec 2940 SCSI 控制器的 CSR 与 NCR 810 SCSI 控制器完全不一样。这些 CSR 被用来启动/停止、初始化设备及对设备进行诊断。

在 Linux 中管理硬件设备控制器的代码并没有放置在每个应用程序中，而是由内核统一管理，这些处理和管理硬件控制器的软件就是设备驱动。

Linux 核心设备驱动是一组运行在特权级上的内存驻留底层硬件处理共享库，负责管理各个设备。

2）设备驱动的基本特征

设备驱动的一个基本特征是设备处理的抽象概念。所有硬件设备都被看成普通文件；可以通过与操纵普通文件相同的标准系统调用来打开、关闭、读取和写入设备。

系统中每个设备都用一种特殊的设备相关文件（device special file）来表示，例如系统中第一个 IDE 硬盘被表示成/dev/hda；块（磁盘）设备和字符设备的设备相关文件可以通过 mknod 命令来创建，并使用主从设备号来描述此设备；网络设备也用设备相关文件来表示，但 Linux 寻找和初始化网络设备时才建立这种文件。

由同一个设备驱动控制的所有设备具有相同的主设备号，从设备号则被用来区分具有相同主设备号且由相同设备驱动控制的不同设备。例如，主 IDE 硬盘的每个分区的从设备号都不相同；/dev/hda2 表示主 IDE 硬盘的主设备号为 3 而从设备号为 2。Linux 通过使用主从设备号将包含在系统调用中的（如将一个文件系统 mount 到一个块设备）设备相关文件映射到设备的设备驱动以及大量系统表格中，例如字符设备表 chrdevs 等。

3）Linux 支持的硬件设备

Linux 支持三类硬件设备：字符、块及网络设备。字符设备指那些无须缓冲直接读写的设备，如系统的串口设备/dev/cua0 和/dev/cua1。块设备则仅能以块为单位读写，典型的块大小为 512B 或 1024B；块设备的存取是通过 Buffer Cache 来进行的，并且可以进行随机访问，即不管块位于设备中何处都可以对其进行读写；块设备可以通过其设备相关文件进行访问，但更为平常的访问方法是通过文件系统；只有块设备才能支持可安装文件系统。网络设备可以通过 BSD 套接口访问。

4）Linux 设备驱动的共性

Linux 核心中虽然存在许多不同的设备驱动，但它们具有一些共性。

（1）核心代码：设备驱动是核心的一部分，像核心中其他代码一样，出错将导致系统的严重损伤，一个编写奇差的设备驱动甚至能使系统崩溃并导致文件系统的破坏和数据丢失。

（2）核心接口：设备驱动必须为 Linux 核心或其从属子系统提供一个标准接口。例如终端驱动为 Linux 核心提供了一个文件 I/O 接口，而 SCSI 设备驱动为 SCSI 子系统提供了一个 SCSI 设备接口，同时此子系统为核心提供了文件 I/O 和 Buffer Cache 接口。

（3）核心机制与服务：设备驱动可以使用标准的核心服务如内存分配、中断发送和等

待队列等。

(4) 动态可加载：多数 Linux 设备驱动可以在核心模块发出加载请求时加载，同时在不再使用时卸载，这样核心能有效地利用系统资源。

(5) 可配置：Linux 设备驱动可以连接到核心中，当核心被编译时，哪些核心被连入核心是可配置的。

(6) 动态性：当系统启动及设备驱动初始化时将查找它所控制的硬件设备。如果某个设备的驱动为一个空过程并不会有什么问题，此时该设备驱动仅仅是一个冗余的程序，除了会占用少量系统内存外不会对系统造成什么危害。

2. 轮询与中断

设备被执行某个命令时，如“将读取磁头移动到软盘的第 42 扇区上”，设备驱动可以从轮询方式和中断方式中选择一种以判断设备是否已经完成此命令。

1) 轮询

轮询方式意味着需要经常读取设备的状态，一直到设备状态表明请求已经完成为止。如果设备驱动被连接进入核心，这时使用轮询方式将会带来灾难性后果：核心将在此过程中无所事事，直到设备完成此请求。但是轮询设备驱动可以通过使用系统定时器，使核心周期性调用设备驱动中的某个例程来检查设备状态。定时器过程可以检查命令状态及 Linux 软盘驱动的工作情况。

使用定时器是轮询方式中最好的一种，但更有效的方法是使用中断。

2) 中断

基于中断的设备驱动会在它所控制的硬件设备需要服务时引发一个硬件中断，例如以太网设备驱动从网络上接收到一个以太数据报时都将引起中断。Linux 核心需要将来自硬件设备的中断传递到相应的设备驱动，这个过程由设备驱动向核心注册其使用的中断来协助完成。此中断处理例程的地址和中断号都将被记录下来。在/proc/interrupts 文件中可以看到设备驱动所对应的中断号及类型：

```
0:      727432    timer
1:      20534     keyboard
2:      0         cascade
3:      79691      + serial
4:      28258      + serial
5:      1         sound blaster
11:     20868      + aic7xxx
13:     1         math error
14:     247        + ide0
15:     170        + ide1
```

对中断资源的请求在驱动初始化时就已经完成。系统中有些中断已经固定，例如软盘控制器总是使用中断 6；其他中断，如 PCI 设备中断，在启动时进行动态分配。设备驱动必须在取得对此中断的所有权之前找到它所控制设备的中断号(IRQ)。Linux 通过支持标准的 PCI BIOS 回调函数来确定系统中 PCI 设备的中断信息，包括其 IRQ 号。

如何将中断发送给 CPU 本身取决于体系结构，但是在多数体系结构中，中断以一种特殊模式发送，同时还将阻止系统中其他中断的产生。设备驱动在其中断处理过程中操作得

越少越好,这样 Linux 核心将能很快地处理完中断并返回中断前的状态中。为了在接收中断时完成大量工作,设备驱动必须能够使用核心的底层处理例程或任务队列来对以后需要调用的那些例程进行排队。

3. 直接内存访问(DMA)

数据量比较少时,使用中断驱动设备驱动程序能顺利地在硬件设备和内存之间交换数据,例如波特率为 9600 的 MODEM 可以每 ms 传输一个字符。如果硬件设备引起中断和调用设备驱动中断所消耗的中断时延比较大(如 2ms),则系统的综合数据传输速率会很低,例如 9600 波特率 MODEM 的数据传输只能利用 0.002%的 CPU 处理时间。高速设备如硬盘控制器或以太网设备数据传速输率将更高,SCSI 设备的数据传输速率可达到每秒 40MB。

直接内存存取(DMA)是解决此类问题的有效方法。DMA 控制器可以在不受处理器干预的情况下在设备和系统内存之间高速传输数据。PC 的 ISA DMA 控制器有 8 个 DMA 通道,其中 7 个可以由设备驱动使用。每个 DMA 通道具有一个 16 位的地址寄存器和一个 16 位的记数寄存器。为了初始化数据传输,设备驱动将设置 DMA 通道地址和记数寄存器以描述数据传输方向以及读写类型,然后通知设备可以在任何时候启动 DMA 操作,传输结束时设备将中断 PC。在传输过程中 CPU 可以转去执行其他任务。

设备驱动使用 DMA 时必须十分小心。首先 DMA 控制器没有任何虚拟内存的概念,它只存取系统中的物理内存;同时用作 DMA 传输缓冲的内存空间必须是连续的物理内存块,这意味着不能在进程虚拟地址空间内直接使用 DMA,但可以将进程的物理页面加锁以防止在 DMA 操作过程中被交换到交换设备上去;另外 DMA 控制器所存取的物理内存有限,因为 DMA 通道地址寄存器代表 DMA 地址的高 16 位,而页面寄存器记录的是其余 8 位,所以 DMA 请求被限制到内存最低 16MB 中。

DMA 通道是非常珍贵的资源,一共才有 7 个,并且还不能在设备驱动间共享。与中断一样,设备驱动必须找到它应该使用的那个 DMA 通道。有些设备使用固定的 DMA 通道,例如软盘设备总使用 DMA 通道 2。有时设备的 DMA 通道可以由跳线来设置,许多以太网设备使用这种技术。设计灵活的设备将告诉系统它将使用哪个 DMA 通道,此时设备驱动仅需要从 DMA 通道中选取即可。

Linux 通过 dma_chan 数组(每个 DMA 通道一个)来跟踪 DMA 通道的使用情况。dma_chan 结构中包含有两个域,一个是指向此 DMA 通道拥有者的指针,另一个指示 DMA 通道是否已经被分配出去。当输入 cat/proc/dma 命令,显示出来的结果就是 dma_chan 结构数组。

4. 内存

设备驱动必须谨慎使用内存。由于它属于核心,所以不能使用虚拟内存。系统接收到中断信号或调度底层任务队列处理过程时,设备驱动将开始运行,而当前进程会发生改变。设备驱动不能依赖于任何运行的特定进程,即使当前是为该进程工作。与核心的其他部分一样,设备驱动使用数据结构来描述它所控制的设备。这些结构被设备驱动代码以静态方式分配,但会增大核心而引起空间的浪费。多数设备驱动使用核心中非页面内存来存储数据。

Linux 为设备驱动提供了一组核心内存分配与回收过程。核心内存以 2 的指数大小的

块来分配，例如 512B 或 128B，此时即使设备驱动的需求小于这个数量也会分配这么多。所以设备驱动的内存分配请求可得到以块大小为边界的内存，这样核心进行空闲块组合更加容易。

请求分配核心内存时，Linux 需要完成许多额外工作。如果系统中空闲内存数量较少，则可能需要丢弃些物理页面或将其写入交换设备。一般情况下 Linux 将挂起请求者并将此进程放置到等待队列中直到系统中有足够的物理内存为止。不是所有的设备驱动（或者真正的 Linux 核心代码）都会经历这个过程，所以若分配核心内存的请求不能立刻得到满足，则此请求可能会失败。如果设备驱动希望在此内存中进行 DMA，那么它必须将此内存设置为 DMA 使能的。这也是为什么是 Linux 核心而不是设备驱动需要了解系统中的 DMA 使能内存的原因。

5. 设备驱动与核心的接口

Linux 核心与设备驱动之间必须有一个以标准方式进行互操作的接口。每一类设备驱动（字符设备、块设备及网络设备）都提供了通用接口以便在需要时为核心提供服务。这种通用接口使得核心可以以相同的方式对待不同的设备及设备驱动，例如 SCSI 和 IDE 硬盘的区别很大，但 Linux 对它们使用相同的接口。

Linux 动态性很强。每次 Linux 核心启动时若遇到不同的物理设备将需要不同的物理设备驱动。Linux 允许通过配置脚本在核心重建时将设备驱动包含在内。设备驱动在启动初始化时可能会发现系统中根本没有任何硬件需要控制。其他设备驱动可以在必要时作为核心模块动态加载。为了处理设备驱动的动态属性，设备驱动在初始化时将其注册到核心中去。Linux 维护着已注册设备驱动表作为与设备驱动的接口，这些表中包含支持此类设备例程的指针和相关信息。

1）字符设备

字符设备是 Linux 设备中最简单的一种。应用程序用可以和存取文件相同的系统调用来打开、读写及关闭它，即使此设备是将 Linux 系统连接到网络中的 PPP 后台进程的 MODEM 也是如此。字符设备初始化时，其设备驱动通过在 device_struct 结构的 chrdevs 数组中添加一个入口来将其注册到 Linux 核心上。设备的主设备标识符用来对此数组进行索引（如对 tty 设备的索引 4）。设备的主设备标识符是固定的。

chrdevs 数组每个入口中的 device_struct 数据结构包含两个元素：一个指向已注册的设备驱动名称，另一个则是指向一组文件操作指针。它们是位于此字符设备驱动内部的文件操作例程的地址指针，用来处理相关的文件操作如打开、读写与关闭。/proc/devices 中字符设备的内容来自 chrdevs 数组。

当打开代表字符设备的字符特殊文件（如/dev/cua0）时，核心必须作好准备以便调用相应字符设备驱动的文件操作例程。与普通的目录和文件一样，每个字符特殊文件用一个 VFS inode 表示。每个字符特殊文件使用的 VFS inode 和所有设备特殊文件一样，包含着设备的主从标识符。这个 VFS inode 由底层的文件系统来建立（例如 ext2），其信息来源于设备相关文件名称所在的文件系统。

每个 VFS inode 和一组文件操作相关联，它们根据 inode 代表的文件系统对象变化而不同。当创建一个代表字符相关文件的 VFS inode 时，其文件操作被设置为默认的字符设备操作。

字符设备只有一个文件操作，即打开文件操作。当应用打开字符特殊文件时，通用文件打开操作使用设备的主标识符来索引此 chrdevs 数组，以便得到那些文件操作函数指针；同时建立起描述此字符特殊文件的 file 结构，使其文件操作指针指向此设备驱动中的文件操作指针集合。这样所有应用对它进行的文件操作都被映射到此字符设备的文件操作集合上。

2）块设备

块设备也支持以文件方式访问。系统对块设备特殊文件提供了非常类似于字符特殊文件的文件操作机制。Linux 在 blkdevs 数组中维护所有已注册的块设备。像 chrdevs 数组一样，blkdevs 也使用设备的主设备号进行索引，其入口也是 device_struct 结构。和字符设备不同的是系统有几类块设备，SCSI 设备是一类，而 IDE 设备则是另外一类，它们将以各自类别登记到 Linux 核心中并为核心提供文件操作功能。某类块设备的设备驱动为此类型设备提供了类别相关的接口，例如 SCSI 设备驱动必须为 SCSI 子系统提供接口以便 SCSI 子系统能用它来为核心提供对此设备的文件操作。

和普通文件操作接口一样，每个块设备驱动必须为 Buffer Cache 提供接口。每个块设备驱动将填充其在 blk_dev 数组中的 blk_dev_struct 结构入口，数组的索引值还是此设备的主设备号。这个 blk_dev_struct 结构包含请求过程的地址以及指向请求数据结构链表的指针，每个代表一个从 Buffer Cache 中来让设备进行数据读写的请求。

每当 Buffer Cache 希望从一个已注册设备中读写数据块时，它会将 request 结构添加到 blk_dev_struc 中。每个请求有指向一个或多个 buffer_hear 结构的指针，每个请求读写一块数据，例如 Buffer Cache 对 buffer_head 结构上锁，则进程会等待到对此缓冲的块操作完成。每个 request 结构都从静态链表 all_requests 中分配。如果此请求被加入到空请求链表中，则将调用驱动请求函数以启动此请求队列的处理，否则该设备驱动将简单地处理请求链表上的 request。

一旦设备驱动完成了请求则它必须将每个 buffer_heard 结构从 request 结构中清除，将它们标识成已更新状态并解锁。对 buffer_head 的解锁将唤醒所有等待此块操作完成的睡眠进程。例如解析文件名称时，ext2 文件系统必须从包含此文件系统的设备中读取包含下个 ext2 目录入口的数据块。在 buffer_head 上睡眠的进程在设备驱动被唤醒后将包含此目录入口，request 数据结构被标识成空闲以便被其他块请求使用。

3）硬盘

磁盘驱动器提供了一个永久性存储数据的方式，将数据保存在旋转的盘片上。写入数据时磁头将磁化盘片上的一个小微粒。这些盘片被连接到一个中轴上并以 3000 到 10 000RPM（每分钟多少转）的恒定速度旋转。而软盘的转速仅为 360RPM。磁盘的读/写磁头负责读写数据，每个盘片的两侧各有一个磁头。磁头读写时并不接触盘片表面而是浮在距表面非常近的空气垫中（百万分之一英寸）。磁头由一个马达驱动在盘片表面移动。所有的磁头被连在一起，它们同时穿过盘片的表面。

盘片的每个表面都被划分成为叫做磁道的狭窄同心圆。0 磁道位于最外面，最大磁道位于最靠近中央主轴。柱面指一组相同磁道号的磁道，例如每个盘片上的第五磁道组成了磁盘的第五柱面。由于柱面号与磁道号相等，所以经常可以看到以柱面描述的磁盘布局。每个磁道可进一步划分成扇区。它是硬盘数据读写的最小单元，同时也是磁盘的块大小。

一般的扇区大小为 512B，并且这个大小可以在磁盘制造出来后格式化时设置。

一个磁盘经常被描述成有多少个柱面、磁头以及扇区。例如系统启动时 Linux 将这样描述一个 IDE 硬盘：

```
hdb:  Conner  Peripherals  540MB - CFS540A,  516MB  w/64kB  Cache,  CHS = 1050/16/63
```

这表示此磁盘有 1050 个柱面（磁道）、16 个磁头（8 个盘片）、每磁道包含 63 个扇区。这样就可以通过扇区数、块数以及 512B 扇区大小计算出磁盘的存储容量为 529 200B。这个容量和磁盘自身声称的 516MB 并不相同，这是因为有些扇区被用来存放磁盘分区信息。有些磁盘还能自动寻找坏扇区并重新索引磁盘以正常使用。

物理硬盘可进一步划分成分区，一个分区是一大组为特殊目的而分配的扇区。对磁盘进行分区使得磁盘可以同时被几个操作系统或不同目的使用。许多 Linux 系统具有 3 个分区：DOS 文件系统分区、ext2 文件系统分区和交换分区。硬盘分区用分区表来描述，表中每个入口用磁头、扇区及柱面号来表示分区的起始与结束。对于用 DOS 格式化的硬盘有 4 个主分区表，但不一定所有的 4 个入口都被使用。fdisk 支持 3 种分区类型：主分区、扩展分区及逻辑分区。扩展分区并不是真正的分区，它只不过包含了几个逻辑分区。扩展分区和逻辑分区用来打破 4 个主分区的限制。以下是一个包含两个主分区的 fdisk 命令的输出：

```
Disk/dev/sda:64 heads, 32 sectors,510 cylinders
Units = cylinders of 2048 * 512 bytes
Device Boot Begin Start End Blocks Id System
/dev/sda1 11   478   489456   83   Linux   native
/dev/sda2   479   479   510   32768   82   Linux   swap
Expert   command   (m   for   help):   p
Disk   /dev/sda:64 heads, 32 sectors,510 cylinders
Nr AF Hd Sec Cyl Hd Sec Cyl Start   Size ID
  100   1   1   0   63   32   477 32   978912   83
  200   0   1   478   63   32   509   978944   65536   82
  300   0   0   0   0   0   0   0   0   00
  400   0   0   0   0   0   0   0   0   00
```

这些内容表明第一个分区从柱面（或者磁道）0、头 1 和扇区 1 开始一直到柱面 477、扇区 22 和头 63 结束。由于每磁道有 32 个扇区、64 个读写磁头，则此分区在大小上等于柱面数。fdisk 使分区在柱面边界上对齐，它从最外面的柱面 0 开始并向中间扩展 478 个柱面。第二个分区（交换分区）从 478 号柱面开始并扩展到磁盘的最内圈。

在初始化过程中，Linux 取得系统中硬盘的拓扑结构映射，它找出有多少种硬盘以及是什么类型，另外还要找到每个硬盘的分区方式。所有这些都用由 gendisk_head 链指针指向的 gendisk 结构链表来表示。每个磁盘子系统如 IDE 在初始化时产生表示磁盘结构的 gendisk 结构；同时它将注册其文件操作例程并将此入口添加到 blk_dev 数据结构中；每个 gendisk 结构包含唯一的主设备号，它与块相关设备的主设备号相同，例如 SCSI 磁盘子系统创建了一个主设备号为 8 的 gendisk 入口，这也是所有 SCSI 硬盘设备的主设备号。

尽管磁盘子系统在其初始化过程中就建立了 gendisk 入口，但是只有 Linux 做分区检

查时才使用。每个磁盘子系统通过维护一组数据结构将物理硬盘上的分区与某个特殊主从特殊设备互相映射。无论何时,通过 Buffer Cache 或文件操作对块设备的读写都将被核心定位到具有某个特定主设备号的设备文件上(如/dev/sda2),而从设备号的定位由各自设备驱动或子系统来映射。

4) IDE 硬盘

Linux 系统上使用得最广泛的硬盘是集成电子磁盘或者 IDE 硬盘。IDE 是一个硬盘接口而不是类似 SCSI 的 I/O 总线接口。每个 IDE 控制器支持两个硬盘,一个为主硬盘,另一个为从硬盘,主、从硬盘可以通过盘上的跳线来设置。系统中的第一个 IDE 控制器成为主 IDE 控制器,而另一个为从属控制器。IDE 可以以每秒 3.3MB 的传输速率传输数据,且最大容量为 538MB。EIDE 或增强式 IDE 可以将磁盘容量扩展到 8.6GB,而数据传输速率为 16.6MB/s。由于 IDE 和 EIDE 都比 SCSI 硬盘便宜,所以大多现代 PC 都包含一个或几个板上 IDE 控制器。

Linux 以其发现控制器的顺序来对 IDE 硬盘进行命名。在主控制器中的主盘为/dev/hda,从盘为/dev/hdb,/dev/hdc 用来表示从属 IDE 控制器中的主盘。IDE 子系统将向 Linux 核心注册 IDE 控制器而不是 IDE 硬盘。主 IDE 控制器的主标识符为 3,从属 IDE 控制器的主标识符为 22。如果系统中包含两个 IDE 控制器,则 IDE 子系统的入口在 blk_dev 和 blkdevs 数组的第 2 和第 22 处。IDE 的块设备文件反映了这种编号方式,硬盘/dev/hda 和/dev/hdb 都连接到主 IDE 控制器上,其主标识符为 3。对 IDE 子系统上这些块相关文件或 Buffer Cache 的操作都通过核心使用主设备标识符作为索引定向到 IDE 子系统上,当发出请求时,此请求由哪个 IDE 硬盘来完成取决于 IDE 子系统。为了做到这一点,IDE 子系统使用从设备编号对应的设备特殊标识符,由它包含的信息将请求发送到正确的硬盘上。位于主 IDE 控制器上的 IDE 从盘/dev/hdb 的设备标识符为(3,64),而此盘中第一个分区(/dev/hdb1)的设备标识符为(3,65)。

6. 初始化 IDE 子系统

IDE 磁盘与 IBM PC 关系非常密切。在这么多年中这些设备的接口发生了变化。这使得 IDE 子系统的初始化过程比看上去要复杂得多。

Linux 可以支持的最多 IDE 控制器个数为 4。每个控制器用 ide_hwifs 数组中的 ide_hwif_t 结构来表示。每个 ide_hwif_t 结构包含两个 ide_drive_t 结构以支持主、从 IDE 驱动器。在 IDE 子系统的初始化过程中,Linux 通过访问系统 CMOS 来判断是否有关于硬盘的信息。这种 CMOS 由电池供电,所以系统断电时也不会遗失其中的内容,它位于永不停止的系统实时时钟设备中。此 CMOS 内存的位置由系统 BIOS 来设置,它将通知 Linux 系统中有多少个 IDE 控制器和驱动器。Linux 使用这些从 BIOS 中发现的磁盘数据来建立对应此驱动器的 ide_hwif_t 结构。

许多现代 PC 系统使用 PCI 芯片组(如 Intel 82430 VX 芯片组)将 PCI EIDE 控制器封装在内。IDE 子系统使用 PCI BIOS 回调函数来定位系统中 PCI (E)IDE 控制器,然后对这些芯片组调用 PCI 特定查询例程。每次找到一个 IDE 接口或控制器就建立一个 ide_hwif_t 结构用来表示控制器和与之相连的硬盘。在操作过程中 IDE 驱动器对 I/O 内存空间中的 IDE 命令寄存器写入命令。主 IDE 控制器的默认控制和状态寄存器地址是 0x1F0～0x1F7,这个地址由早期的 IBM PC 规范设定。IDE 驱动器为每个控制器向 Linux 注册块缓

冲 Cache 和 VFS inode,并将其加入到 blkdev 和 blkdevs 数组中。IDE 驱动器需要申请某个中断,一般主 IDE 控制器的中断号为 14,而从属 IDE 控制器的为 15。这些都可以通过命令行选项由核心来重载。IDE 驱动器同时还将 gendisk 入口加入到启动时发现的每个 IDE 控制器的 gendisk 链表中去。分区检查代码知道每个 IDE 控制器可能包含两个 IDE 硬盘。

7. SCSI 设备

SCSI(小型计算机系统接口)总线是一种高效的点对点数据总线,它最多可以支持 8 个设备,其中包括多个主设备。每个设备有唯一的标识符并可以通过盘上的跳线来设置。在总线上的两个设备间,可以以同步或异步方式在 32 位数据宽度下传输速率为 40MB/s 来交换数据。SCSI 总线上可以在设备间同时传输数据与状态信息。initiator 设备和 target 设备间的执行步骤最多可以包括 8 个不同的阶段,可以从总线上的信号来分辨 SCSI 总线的当前阶段。这 8 个阶段是:

(1) BUS FREE。当前没有设备在控制总线且总线上无事务发生。

(2) ARBITRATION。一个 SCSI 设备试图取得 SCSI 总线的控制权,这时它将其 SCSI 标识符放置到地址引脚上。具有最高 SCSI 标识符编号的设备将获得总线控制权。

(3) SELECTION。当设备通过仲裁成功地取得了对 SCSI 总线的控制权后,必须向它准备发送命令的那个 SCSI 设备发出信号。具体做法是将目标设备的 SCSI 标识符放置在地址引脚上进行声明。

(4) RESELECTION。在一个请求的处理过程中,SCSI 设备可能会断开连接,目标(target)设备将再次选择启动设备(initiator)。不是所有的 SCSI 设备都支持此阶段。

(5) COMMAND。此阶段中 initiator 设备将向 target 设备发送 6B、10B 或 12B 命令。

(6) DATA IN,DATA OUT。此阶段中数据将在 initiator 设备和 target 设备间传输。

(7) STATUS。所有命令完毕后将进入此阶段,此时允许 target 设备向 initiator 设备发送状态信息以指示操作成功与否。

(8) MESSAGE IN,MESSAGE OUT。此阶段附加信息将在 initiator 设备和 target 设备间传输。

Linux SCSI 子系统由两个基本部分组成,每个由一个数据结构来表示。

(1) host。一个 SCSI host 即一个硬件设备:SCSI 控制权。NCR 810 PCI SCSI 控制权即一种 SCSI host。在 Linux 系统中可以存在相同类型的多个 SCSI 控制权,每个由一个单独的 SCSI host 来表示,这意味着一个 SCSI 设备驱动可以控制多个控制权实例。SCSI host 总是 SCSI 命令的 initiator 设备。

(2) Device。虽然 SCSI 支持多种类型设备如磁带机、CD-ROM 等,但最常见的 SCSI 设备是 SCSI 磁盘。SCSI 设备总是 SCSI 命令的 target 设备。这些设备必须区别对待,例如 CD-ROM 或磁带机这种可移动设备,Linux 必须检测介质是否已经移动。不同的磁盘类型有不同的主设备号,这样 Linux 可以将块设备请求发送到正确的 SCSI 设备。

8. 初始化 SCSI 子系统

SCSI 子系统的初始化非常复杂,它必须反映出 SCSI 总线及其设备的动态性。Linux 在启动时初始化 SCSI 子系统,如果它找到一个 SCSI 控制器(即 SCSI host)则会扫描此 SCSI 总线来找出总线上的所有设备,然后初始化这些设备并通过普通文件和 Buffer Cache 块设备操作使 Linux 核心的其他部分能使用这些设备。初始化过程分成 4 个阶段:

(1) 首先 Linux 将找出在系统核心连接时,被连入核心的哪种类型的 SCSI 主机适配器或控制器有硬件需要控制。每个核心中的 SCSI host builtin_scsi_hosts 数组中有一个 Scsi_Host_Template 入口。而 Scsi_Host_Template 结构中包含执行特定 SCSI host 操作,如检测连到此 SCSI host 的 SCSI 设备的例程的入口指针。这些例程在 SCSI 子系统进行自我配置时使用,同时它们还是支持此 host 类型的 SCSI 设备驱动的一部分。每个被检测的 SCSI host,即与真正 SCSI 设备连接的 host 将其自身的 Scsi_Host_Template 结构添加到活动 SCSI hosts 的 scsi_hosts 结构链表中去。每个被检测 host 类型的实例用一个 scsi_hostlist 链表中的 Scsi_Host 结构来表示。例如一个包含两个 NCR 810 PCI SCSI 控制器的系统的链表中将有两个 Scsi_Host 入口,每个控制器对应一个。每个 Scsi_Host 指向一个代表设备驱动的 Scsi_Host_Template。

(2) 现在每个 SCSI host 已经找到,SCSI 子系统必须找出哪些 SCSI 设备连接哪个 host 的总线。SCSI 设备的编号是从 0 到 7,对于一条 SCSI 总线上连接的各个设备,其设备编号或 SCSI 标识符是唯一的。SCSI 标识符可以通过设备上的跳线来设置。SCSI 初始化代码通过在 SCSI 总线上发送一个 TEST_UNIT_READY 命令来找出每个 SCSI 设备。当设备做出响应时其标识符通过一个 ENQUIRY 命令来读取。Linux 将从中得到生产厂商的名称和设备模式以及修订版本号。SCSI 命令由一个 Scsi_Cmnd 结构来表示,同时这些命令通过调用 Scsi_Host_Template 结构中的设备驱动例程传递到此 SCSI host 的设备驱动中。被找到的每个 SCSI 设备用一个 Scsi_Device 结构来表示,每个指向其父 Scsi_Host 结构。所有这些 Scsi_Device 结构被添加到 scsi_device 链表中。

(3) 一共有 4 种 SCSI 设备类型:磁盘、磁带机、CD-ROM 和普通 SCSI 设备。每种类型的 SCSI 设备以不同的主块设备类型单独登记到核心中。如果有多个类型的 SCSI 设备存在,则它们只登记自身。每个 SCSI 设备类型(如 SCSI 磁盘)维护着其自身的设备列表,它使用这些表将核心块操作(file 或者 Buffer Cache)定向到正确的设备驱动或 SCSI host 上。每种 SCSI 设备类型用一个 Scsi_Device_Template 结构来表示,此结构中包含此类型 SCSI 设备的信息以及执行各种任务的例程的入口地址。换句话说,如果 SCSI 子系统希望连接一个 SCSI 磁盘设备,它将调用 SCSI 磁盘类型连接例程。如果有多个该种类型的 SCSI 设备被检测到,则此 Scsi_Type_Template 结构将被添加到 scsi_devicelist 链表中。

(4) 初始化 SCSI 子系统的最后一个阶段是为每个已登记的 Scsi_Device_Template 结构调用 finish 函数。对于 SCSI 磁盘类型设备,它将驱动所有 SCSI 磁盘并记录其磁盘布局。

一旦 SCSI 子系统初始化完成,这些 SCSI 设备就可以使用了。每个活动的 SCSI 设备类型将其自身登记到核心以便 Linux 正确定向块设备请求。这些请求可以是通过 blk_dev 的 Buffer Cache 请求,也可以是通过 blkdevs 的文件操作。以一个包含多个 ext2 文件系统分区的 SCSI 磁盘驱动器为例,当安装其中一个 ext2 分区时,系统将核心缓冲请求定向到正确的 SCSI 磁盘的过程如下:

每个对 SCSI 磁盘分区的块读写请求将导致一个新的 request 结构被添加到对应此 SCSI 磁盘的 blk_dev 数组中的 current_request 链表中。如果此 request 正在被处理,则 Buffer Cache 无须做任何工作;否则它必须通知 SCSI 磁盘子系统去处理它的请求队列。系统中每个 SCSI 磁盘用一个 Scsi_Disk 结构来表示,例如/dev/sdb1 的主设备号为 8 而从设备号为 17,这样产生一个索引值 1。每个 Scsi_Disk 结构包含一个指向表示此设备的

Scsi_Device 结构。这样反过来又指向拥有它的 Scsi_Host 结果。这个来自 Buffer Cache 的 request 结构将被转换成一个描述 SCSI 命令的 Scsi_Cmd 结构，这个 SCSI 命令将发送到此 SCSI 设备同时被排入表示此设备的 Scsi_Host 结构。一旦有适当的数据块需要读写，这些请求将被独立的 SCSI 设备驱动来处理。

9. 网络设备

网络设备即 Linux 的网络子系统，是一个发送与接收数据包的实体。它一般是一个类似以太网卡的物理设备。有些网络设备如 loopback 设备仅仅是一个用来向自身发送数据的软件。每个网络设备都用一个 device 结构来表示。网络设备驱动在核心启动初始化网络时将这些受控设备登记到 Linux 中。device 数据结构中包含有关设备的信息以及用来支持各种网络协议的函数地址指针，这些函数主要用来使用网络设备传输数据。设备使用标准网络支持机制来将接收到的数据传输到适当的协议层。所有传输与接收到的网络数据用一个 sk_buff 结构来表示，这些灵活的数据结构使得网络协议头可以更容易地添加和删除。

device 数据结构包含以下有关网络设备的信息：

1）Name

与使用 mknod 命令创建的块设备特殊文件与字符设备特殊文件不同，网络设备特殊文件仅在于系统网络设备发现与初始化时建立。它们使用标准的命名方法，每个名字代表一种类型的设备。多个相同类型设备将从 0 开始记数。这样以太网设备被命名为/dev/eth0、/dev/eth1、/dev/eth2 等。一些常见的网络设备名称如下：

(1) /dev/eth*N*：以太网设备。

(2) /dev/sl*N*：SLIP 设备。

(3) /dev/ppp*N*：PPP 设备。

(4) /dev/lo：Loopback 设备。

2）Bus Information

这些信息被设备驱动用来控制设备。irq 号表示设备使用的中断号；base address 指任何设备在 I/O 内存中的控制与状态寄存器地址；DMA 通道指此网络设备使用的 DMA 通道号。所有这些信息在设备初始化时设置。

3）Interface Flags

描述网络设备的属性与功能。

(1) IFF_UP：接口已经建立并运行。

(2) IFF_BROADCAST：设备中的广播地址有效。

(3) IFF_DEBUG：设备调试被使能。

(4) IFF_LOOPBACK：这是一个 loopback 设备。

(5) IFF_POINTTOPOINT：这是点到点连接(SLIP 和 PPP)。

(6) IFF_NOTRAILERS：无网络追踪者。

(7) IFF_RUNNING：资源已被分配。

(8) IFF_NOARP：不支持 ARP 协议。

(9) IFF_PROMISC：设备处于混乱的接收模式，无论包地址怎样它都将接收。

(10) IFF_ALLMULTI：接收所有的 IP 多播帧。

(11) IFF_MULTICAST：可以接收 IP 多播帧。

4) Protocol Information

每个设备描述其可以被网络协议层如何使用。

(1) mtu：指不包括任何链路层头在内的、网络可传输的最大包大小。这个值被协议层用来选择适当大小的包进行发送。

(2) Family：这个 Family 域表示设备支持的协议族。所有 Linux 网络设备的族是 AF_INET，Internet 地址族。

(3) Type：这个硬件接口类型描述网络设备连接的介质类型。Linux 网络设备可以支持多种不同类型的介质，包括以太网、X.25、令牌环、Slip、PPP 和 Apple Localtalk。

(4) Addresses：结构中包含大量网络设备相关的地址，包括 IP 地址。

(5) Packet Queue：指网络设备上等待传输的 sk_buff 包队列。

(6) Support Functions：每个设备支持一组标准的例程，它们被协议层作为设备链路层。

3.3 Android 内核深度解析

3.3.1 Android 内核分析

1. 内核在操作系统中的地位

Android 是基于 Linux 操作系统的，由硬件、系统内核、系统服务和应用程序四大部分组成。内核(Kernel)是最核心的部分，其主要作用在于与计算机硬件进行交互，实现对硬件的编程控制和接口操作，调度访问硬件资源，同时向应用程序提供一个高级的执行环境和对硬件的虚拟接口。内核的主要功能包括：

(1) 中断服务程序。

(2) 进程调度程序。

(3) 进程地址空间的内存管理。

(4) 进程间通信。

内核与普通应用程序不同，其拥有所有硬件设备的访问权限以及启动时即划分的受保护的内存空间。

2. Android 内核简介

和标准的 Linux 内核一样，Android 内核主要实现内存管理、进程调度、进程间通信等功能。

Android 内核基于 linux 2.6 内核，提供安全、内存管理、进程管理、网络组和驱动模型等核心服务，同所有 Linux 内核一样，Android 内核是介于硬件层和软件组之间的一个抽象层次。为了适应嵌入式硬件环境和移动应用程序的开发，Android 对标准 Linux 内核进行了一定的修改。为了对比分析 Android 内核，在 Ubuntu 操作系统上搭建了 Android 内核的编译开发平台，通过 repo 下载最新的 Android 内核代码版本 cupcake。从获得的内核源码树的根目录结构看，Android 内核源码与标准 Linux 内核并无不同，如表 3-1 所示。

表 3-1　Android 内核源码树根目录结构表

目　　录	描　　述	目　　录	描　　述
Arch	特定体系结构的源码	Init	内核引导和初始化
Crypto	Crypto API	Ipc	进程间通信代码
Documentation	内核源码文档	Lib	通用内核函数
Drivers	设备驱动程序	Mm	内存管理模块
Fs	VFS 和各种文件系统	Net	网络模块
Include	内核头文件	Scripts	编译内核所用的脚本

3. Android 内核与 Linux 内核的区别

Android 内核的结构和标准的 Linux 内核是基本相同的，Android 在 Linux 内核基础上增加了私有内容，主要是一些驱动程序。这些驱动程序主要分成两种类型，一种是 Android 专用驱动程序，另一种是 Android 使用的设备驱动程序。

经过与标准 Linux 内核源代码进行详细对比还可以发现，Android 内核与标准 Linux 内核在文件系统、进程间通信机制、内存管理、电源管理等许多方面也存在不同。

1）文件系统

不同于桌面系统与服务器，移动设备大多采用的不是硬盘而是采用 Flash 作为存储介质，因此 Android 内核中增加了标准 Linux 内核中没有采纳的 YAFFS2 文件系统。YAFFS2 是专用于 Flash 的文件系统，对 NAND Flash 芯片有着良好的支持；它是日志结构的文件系统，提供了损耗平衡和掉电保护，可以有效地避免意外断电对文件系统一致性和完整性的影响。YAFFS2 文件系统按层次结构设计，分为文件管理接口、内部实现层和 NAND，简化了其本身与系统的接口设计，能更方便地集成到系统当中。经过测试证明，YAFFS2 文件系统性能比支持 NOR 型闪存的 JFFS2 文件系统优秀。

2）进程间通信机制

Android 增加了一种进程间的通信机制 IPC Binder，在内核源代码中驱动程序文件为 coredroid/include/linux/binder.h 和 coredroid/drivers/android/binder.c。Binder 通过守护进程 Service Manager 管理系统中的服务，负责进程间的数据交换。各进程通过 Binder 访问同一块共享内存以达到数据通信的机制。从应用层的角度看，进程通过访问数据守护进程获取用于数据交换的程序框架接口，调用并通过接口共享数据，而其他进程要访问数据也只需与程序框架接口进行交互，这方便了程序员开发需要交互数据的应用程序。

3）内存管理

在内存管理模块，Android 内核采用了一种不同于标准 Linux 内核的低内存管理策略。在标准 Linux 内核当中，使用一种叫做 OOM 的低内存管理策略，当内存不足时，系统检查所有的进程，并对进程进行限制评分，获得最高分的进程将被关闭。Android 系统采用的则是一种叫做 LMK 的机制，这种机制将进程按照重要性进行分级、分组。内存不足时，将处于最低级别组的进程关闭。例如在移动设备当中，UI 界面处于最高级别，所以该进程永远不会被中止，这样在终端用户看来系统是稳定运行的。在 Android 内核源码中，LMK 的位置是 coredroid/drivers/misc/lowme -morykiller.c。

与此同时，Android 新增加了一种内存共享的处理方式 Ashmem，即匿名共享内存。通过 Ashmem，进程间可以匿名自由共享具名的内存块，这种共享方式在标准 Linux 当中不被

支持。在 Andorid 内核源码中，Ashmem 的位置是 coredroid/mm/ashmem.c。

4）电源管理

由于 Android 主要用于移动设备，电源管理就显得尤为重要，因此在 Android 内核当中增加了一种新的电源管理策略。目前 Android 采用的是一种较为简单的电源管理策略，通过开关屏幕、开关屏幕背光、开关键盘背光、开关按钮背光和调整屏幕亮度来实现电源管理，并没有实现休眠和待机功能。

有 3 种途径判断调整电源管理策略：RPC 调用、电池状态改变和电源设置。它通过广播 Intent 或直接调用 API 的方式来与其他模块进行联系。电源管理策略同时还有自动关机机制，当电力低于最低可接受程度时系统将自动关机。

Android 的电源管理模块还会根据用户行为自动调整屏幕亮度，其响应机制如图 3-4 所示。

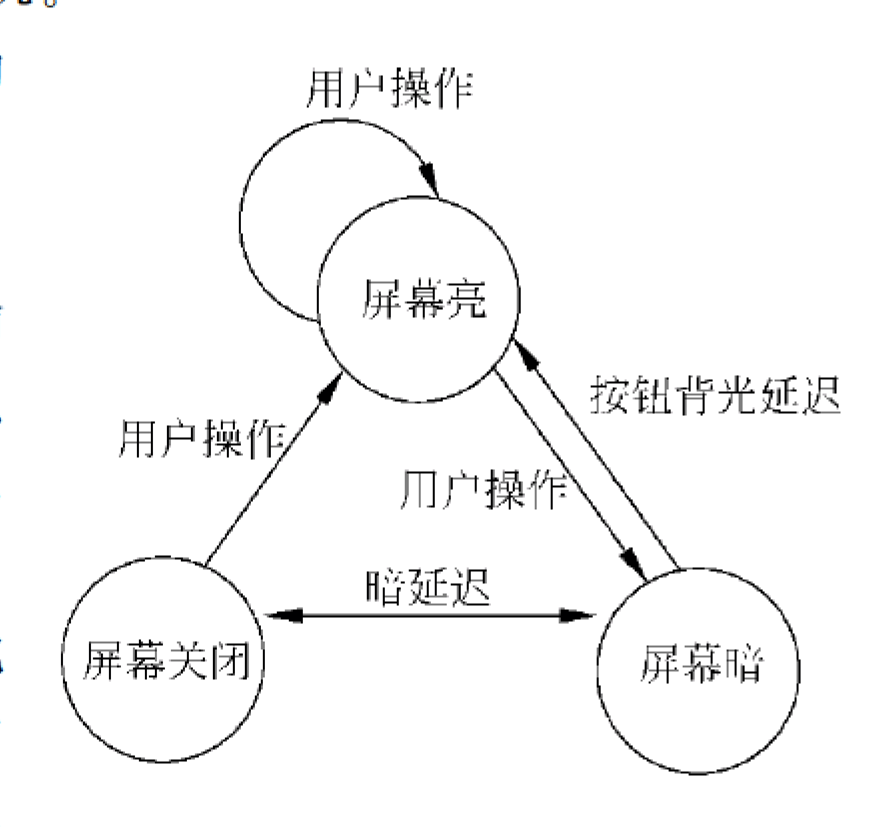


图 3-4　屏幕背光响应机制

5）驱动及其他

相对于标准内核，Android 内核还添加了字符输出设备、图像显示设备、键盘输入设备、RTC 设备、USB Device 设备等相关设备驱动；增加了日志系统，使应用程序可以访问日志消息。

Android 内核由标准 Linux 内核修改而来，因此继承了 Linux 内核的各种优点，保留了标准 Linux 内核的主体结构；同时 Android 按照移动设备的需求，在以上介绍的文件系统、内存管理、进程间通信机制、电源管理等方面进行了修改，还添加了相关的驱动程序和一些必要的新功能。因此 Android 系统应用范围更加广泛，拓展性更强。

4．Android 内核获取与编译

以 Ubuntu 9.0.4 操作系统为例，在 Ubuntu 上搭建 Android 内核编译环境，具体步骤如下：

（1）准备系统环境，需要的软件包为 flex、bison、gperf、libsdl-dev、libesd0-dev、libwxgtk2.6-dev(optional)、build-essential、zip 和 curl。

（2）安装 JDK 的 1.5 版本。

（3）安装 repo。

（4）在主文件目录建立一个 bin 文件夹并加入到环境变量当中。

（5）下载 repo 脚本，并将它的属性改为可执行 curl。下载地址为：

```
http://android.git.kernel.org/repo > ～/bin/repo chmoda + x ～/bin/repo
```

（6）创建一个保存源码的目录，命令为：

```
mkdir coredroid
cd coredroid
```

（7）获取 Android 内核代码：

```
git clone git://android.git.kernel.org/kernel/common.git
```

经过以上步骤，最新版本 Android 内核就被下载到文件夹 coredroid 当中了。

（8）编译：

```
cd cordroid
make xconfig      ＃可按 doucument 文件夹下的 android.txt 说明进行配置
make              ＃编译完成后生成 zImage 文件，可用 Emulator 测试运行
cd mydroid/out/cupcake/out/target/product/generic
emulator - image system.img - data userdata.img - ramdisk
ramdisk.img - kernel ～/coredroid/common /arch/arm/boot/zImage
```

3.3.2 Android 内核剖析

1. Binder 框架

Binder 是一种结构，这种结构提供了服务端、Binder 驱动、客户端 3 个模块。

1）服务端

一个 Binder 服务端实际就是一个 Binder 类的对象，该对象一旦创建，内部就启动一个隐藏线程。该线程接下来会接收 Binder 驱动发送的消息，收到消息后会执行到 Binder 对象中的函数，并按照该函数的参数执行不同的服务代码。因此，要实现一个 Binder 服务就必须重载 onTransact()函数。可以想象，重载 onTransact()函数的主要内容是把 onTransact()函数的参数转换为服务函数的参数，而 onTransact()函数的参数来源是客户端调用 transact()函数时输入的。因此，如果 transact()有固定格式的输入，那么 onTransact()就会有固定格式的输出。

2）Binder 驱动

任意一个服务端 Binder 对象被创建时，同时会在 Binder 驱动中创建一个对象，该对象的类型也是 Binder 类。客户端要访问远程服务时都是通过这个对象。

3）客户端

最后来看应用程序客户端。客户端要想访问远程服务，必须获取远程服务在 Binder 对象中对应的 mRemote 引用。获得该 mRemote 对象后就可以调用其 transact()函数。而在 Binder 驱动中，mRemote 对象也重载了 transact()函数，重载的内容主要包括以下几项：

（1）以线程间消息通信的模式向服务端发送客户端传递过来的参数。

（2）挂起当前线程，当前线程正是客户端线程，并等待服务端线程执行完指定服务函数后的通知(notify)。

（3）接收到服务端线程的通知，然后继续执行客户端线程，并返回到客户端代码区。

对应用程序开发员来讲，客户端似乎是直接调用远程服务对应的 Binder，而事实上则是通过 Binder 驱动进行了中转。即存在两个 Binder 对象，一个是服务端的 Binder 对象，另一个则是 Binder 驱动中的 Binder 对象，所不同的是 Binder 驱动中的对象不会再额外产生一个线程。

2. Framework 概述

Framework 定义了客户端组件和服务端组件功能及接口。其框架中包含 3 个主要部分：服务端、客户端和 Linux 驱动。

1）服务端

服务端主要包含两个重要类，分别是 WindowManagerService（WmS）和 ActivityManagerService(AmS)。前者的作用是为所有的应用程序分配窗口，并管理这些窗口，包括分配窗口的大小、调节各窗口的叠放次序、隐藏或者显示窗口；后者的作用是管理所有应用程序中的 Activity。

除此之外，服务端还包括两个消息处理类。

(1) KeyQ 类：为 WmS 的内部类，继承于 KeyInputQueue 类。keyQ 对象一旦创建就立即启动一个线程，该线程会不断地读取用户的 UI 操作消息，例如按键、触摸屏、trackball、鼠标等，并把这些消息放到一个消息队列 QueueEvent 类中。

(2) InputDispatcherThread：该类的对象一旦创建也会立即启动一个线程，该线程会不断地从 QueueEvent 中取出用户消息，并进行一定的过滤，过滤后再将这些消息发给当前活动的客户端程序中。

2）客户端

客户端主要包括以下重要类。

(1) ActivityThread 类：该类为应用程序的主线程类，所有的 APK 程序都有且仅有一个 ActivityThread 类，程序的入口为该类中的 static main()函数。

(2) Activity 类：该类为 APK 程序的一个最小运行单元，一个 APK 程序中可以包含多个 Activity 对象，ActivityThread 主类会根据用户操作选择运行哪个 Activity 对象。

(3) PhoneWindow 类：该类继承于 Window 类，同时 PhoneWindow 类内部包含了一个 DecorView 对象。简而言之，PhoneWindow 类把一个 FrameLayout 进行了一定的包装，并提供了一组通用的窗口操作接口。

(4) Window 类：该类提供了一组通用的窗口操作 API。这里的窗口仅仅是程序层面上的。WmS 所管理的窗口并不是 Window 类，而是一个 View 或者 ViewGroup 类，一般是指 DecorView 类，即一个 DecorView 就是 WmS 所管理的一个窗口。

(5) DecorView 类：该类是一个 FrameLayout 的子类，并且是 PhoneWindow 类中的一个内部类。DecorView 类就是对普通的 FrameLayout 进行了一定的修饰。

(6) ViewRoot 类：WmS 管理客户端窗口时，需要通知客户端进行某种操作，这些都是通过异步消息完成的，其作用主要是接收 WmS 的通知。

(7) W 类：该类继承于 Binder，并且是 ViewRoot 类的一个内部类。

(8) WindowManager 类：客户端要申请创建一个窗口，而具体创建窗口的任务是由 WmS 完成的，客户端不能直接和 WmS 进行交互。

Linux 驱动和 Framework 相关的部分主要有两个，分别是 SurfaceFlingger(SF)和 Binder。每个窗口都对应一个 Surface，SF 驱动的作用是把各个 Surface 显示在同一个屏幕上。Binder 驱动的作用是提供跨进程的消息传递。

3）APK 程序的运行过程

(1) ActivityThread 从 main()函数中开始执行，调用 prepareMainLooper()为 UI 线程创建一个消息队列(MessageQueue)。

(2) 创建一个 ActivityThread 对象，在 ActivityThread 的初始化代码中会创建一个 H (Handler)对象和一个 ApplicationThread(Binder)对象。其中 Binder 负责接收远程 AmS

的 IPC 调用,接收到调用后则通过 Handler 把消息发送到消息队列,UI 主线程会异步地从消息队列中取出消息并执行相应操作,例如 start、stop、pause 等。

(3) UI 主线程调用 Looper.loop()方法进入消息循环体,进入后就会不断地从消息队列中读取并处理消息。

(4) 当 ActivityThread 接收到 AmS 发送 start 某个 Activity 后,就会创建指定的 Activity 对象。Activity 又会创建 PhoneWindow 类→DecorView 类→创建相应的 View 或者 ViewGroup。创建完成后,Activity 需要把创建好的界面显示到屏幕上,于是调用 WindowManager 类,后者于是创建一个 ViewRoot 对象。该对象实际上创建了 ViewRoot 类和 W 类。创建 ViewRoot 对象后,WindowManager 再调用 WmS 提供的远程接口,完成添加一个窗口并显示到屏幕上。

(5) 接下来,用户开始在程序界面上操作。KeyQ 线程不断把用户消息存储到 QueueEvent 队列中,InputDispatcherThread 线程逐个取出消息,然后调用 WmS 中的相应函数处理该消息。当 WmS 发现该消息属于客户端某个窗口时,就会调用相应窗口的 W 接口。

W 类是一个 Binder,负责接收 WmS 的 IPC 调用,并把调用消息传递给 ViewRoot 类,ViewRoot 类再把消息传递给 UI 主线程 ActivityThread,ActivityThread 解析该消息并做相应的处理。在客户端程序中,首先处理消息的是 DecorView 类。如果 DecorView 类不想处理某个消息,则可以将该消息传递给其内部包含的子 View 或者 ViewGroup,如果还没有处理则传递给 PhoneWindow 类,最后再传递给 Activity 类。

4) Framework 的启动过程

系统中运行的第一个 Dalvik 虚拟机程序叫做 Zygote,包含以下两个主要模块:

(1) Socket 服务端。该 Socket 服务端用于接收启动新的 Dalvik 进程的命令。

(2) Framework 共享类和共享资源。当 Zygote 进程启动后,会装载一些共享的类及资源,其中共享类是在 preload-classes 文件中被定义,共享资源是在 preload-resources 中被定义。因此,这些类和资源装载后,新的 Dalvik 进程就不需要再装载这些类和资源了,这就是所谓的共享。

folk 是 Linux 系统的一个系统调用,其作用是复制当前进程,产生一个新的进程。新进程将拥有和原始进程完全相同的进程信息,除了进程 ID 不同。进程信息包括该进程所打开的文件描述符列表、所分配的内存等。当新进程创建后,两个进程将共享已经分配的内存空间,直到其中一个需要向内存中写入数据时,操作系统才负责复制一份目标地址空间,并将要写的数据写入到新的地址中。这就是所谓的 copy-on-write 机制,即"仅当写的时候才复制",这种机制可以最大限度地在多个进程中共享物理内存。

在所有的操作系统中都存在一个程序装载器。程序装载器一般会作为操作系统的一部分,并由所谓的 Shell 程序调用。当内核启动后,Shell 程序会首先启动。常见的 Shell 程序包含两大类,一类是命令行界面,另一类是窗口界面。Windows 系统中 Shell 程序就是桌面程序,Ubuntu 系统中的 Shell 程序就是 GNOME 桌面程序。Shell 程序启动后,用户可以双击桌面图标启动指定的应用程序。而在操作系统内部,启动新的进程包含以下 3 个过程:

(1) 内核创建一个进程数据结构,用于表示将要启动的进程。

(2) 内核调用程序装载器函数,从指定的程序文件读取程序代码,并将这些程序代码装载到预先设定的内存地址。

(3) 装载完毕后,内核将程序指针指向到目标程序地址的入口处开始执行指定的进程。当然,实际的过程会考虑更多的细节,不过大致思路就是这么简单。

在一般情况下,没有必要复制进程,而是按照以上 3 个过程创建新进程。但当满足以下条件时,则建议使用复制进程:两个进程中共享了大量的程序。从系统结构的角度来讲,先创建一个 Zygote,并加载共享类和资源,然后通过该 Zygote 去孵化新的 Dalvik 进程。该结构的特点是:

(1) 每一个进程都是一个 Dalvik 虚拟机,而 Dalvik 虚拟机是一种类似于 JVM 的程序。

(2) Zygote 进程预先会装载共享类和共享资源,这些类及资源实际是 SDK 中定义的大部分类和资源。

Framework 启动时需要加载运行两个特定的 Java 类,一个是 ZygoteInit. java,一个是 SystemServer. java。

Dalvik 虚拟机相关可执行程序的名称和源码路径如表 3-2 所示。

表 3-2　Dalvik 虚拟机相关可执行程序的名称和源码路径

名　称	源码路径
Dalvikvm	Dalvik/dalvikvm
Dvz	Dalvik/dvz
App_process	Frameworks/base/cmds/app_process

AmS 的启动模式如下:

(1) 调用 main()函数,返回一个 Context 对象而不是 AmS 服务本身。

(2) 调用 AmS. setSystemProcess()函数。

(3) 调用 AmS. installProviders()函数。

(4) 调用 systemReady()函数。当 AmS 执行完 systemReady()函数后,会相继启动相关联服务的 systemReady()函数,完成整体初始化。

3. AmS 内部原理

ActivityManagerService. java 文件,简称 AmS,是 Android 内核的三大核心功能之一。

1) AmS 主要提供的功能

(1) 统一调度各应用程序的 Activity。应用程序要运行 Activity 会首先报告给 AmS,然后由 AmS 决定该 Activity 是否可以启动。

(2) 内存管理。Activity 退出后,其所在的进程并不会被立即杀死,从而在下次启动该 Activity 时能够提高启动速度。这些 Activity 只有当系统内存紧张时才会被自动杀死。

(3) 进程管理。AmS 向外提供了查询系统正在运行的进程信息的 API。

2) 统一调度应用程序的 Activity

Activity 本身只是一段程序代码而已,它所执行的内容没有任何的系统调用,客户端程序总是从 ActivityThread 类开始执行。这个类已经创建了客户进程,然后从 Activity 所对应的 Class 文件中装载程序代码,实际上就只是一个 Activity 类对象,然后继续执行该对象内部的各种代码,这些执行的代码默认不会再创建线程。

Activity 并不对应一个应用程序，ActivityThread 类才对应一个应用程序，因此 Android 允许同时运行多个应用程序实际上是允许多个 ActivityThread 类同时运行。默认的 Activity 实现中的确会添加一个窗口，但实际上可以修改 Activity 的默认实现使其不添加任何窗口，而且 AmS 对这种修改不会在意，依然会正常运行。换句话说，AmS 会按照定义好的调度顺序来启动、关闭 Activity，不在意 Activity 内部是否添加窗口。在 Android 中，尽管默认的 Activity 会添加一个窗口，但是当系统内存低的时候，后台 Activity 对应的显存会被释放掉，而且这是在用户没有感知的情况下进行的。Android 窗口系统是一种简单的单窗口系统。

Activity 调度机制的基本思路是：各应用进程要启动新的 Activity 或者停止当前的 Activity 都要先报告给 AmS。AmS 在内部为所有应用进程做记录，当接到启动或停止报告时，首先更新内部记录，然后再通知相应客户进程运行或者停止指定的 Activity。具体地，启动一个 Activity 有以下几种方式：

（1）在应用程序中调用 startActivity()启动指定的 Activity。

（2）在 Home 程序中单击一个应用图标启动新的 Activity。

（3）按 Back 键，结束当前 Activity，自动启动上一个 Activity。

（4）长按 Home 键，显示当前任务列表，从中选择一个启动。

3）内存管理

Android 中的内存管理分为两种情况：

（1）当应用程序关闭后，后台对应的进程并没有真正退出，以便下次在启动时能够快速启动。“关闭”仅仅是使其对应的窗口不显示，而对应的进程会一直保存。这种机制仅仅占用内存，基本不会降低前台程序的运行速度。

（2）当系统内存不够用时，AmS 会主动根据一定的优先规则退出优先级较低的进程。

关闭程序的所有进程包含以下 3 种情况：

（1）从调用 startActivity()方法开始，在一般情况下当前都有正在运行的 Activity，而暂停完毕后 AmS 会收到一个 Binder 消息，并开始从 completePause()方法处执行；在该方法中，由于上一个 Activity 并没有 finishing，仅仅是 stop，所以这会把上一个 Activity 添加到 mStoppingActivities 列表中；当目标 Activity 启动后会向 AmS 发送一个请求进行内存回收的消息，这会导致 AmS 在内部调用 ActivityIdleInternal()方法；该方法中首先会处理 mStoppingActivities 列表中的 Activity 对象，当 stop 完毕后再报告给 AmS，于是 AmS 再从 activityStopped()方法处开始执行；而这会调用 trimApplications()方法，该方法中则会执行和内存相关的操作。

（2）当按 Back 键后，会调用 finishActivityLocked()方法，然后把该 Activity 的 finishing 标识设为 true；然后再调用 startPausingLocked()方法；当目标 Activity 完成暂停后就会报告给 AmS，此时 AmS 又会从 completePause()方法处开始执行。与第一种情况不同，由于此时暂停的 Activity 的 finishing 状态已经设置为 true，所以会执行 finishActivityLocked()方法。

（3）当 Activity 启动后，会向 AmS 发送一个 Idle 消息，这会导致 AmS 开始执行 activityIdleInternal()方法；该方法中会首先处理 mStoppingActivities 列表中的对象，接着处理 mFinishingActivities 列表，最后再调用 trimApplication()方法。

4）进程管理

从内存回收的角度来看，释放内存的地点包含 3 个：

（1）在 AmS 中进行。即 Android 所声称的当系统内存低时优先释放没有任何 Activity 的进程，然后释放非前台 Activity 对应的进程。

（2）在 OOMKiller 中。此时 AmS 只要告诉 OOM 各个应用进程的优先级，然后 OOM 就会调用 Linux 内部的进程管理方法杀死优先级较低的进程。

（3）在应用程序本身之中。当 AmS 任务目标进程需要被杀死时，首先肯定会通知目标进程进行内存释放，这包括调用目标进程的 scheduleLowMemory()方法和 processInBackground()方法。

系统按照以下优先级关闭进程以释放内存，程序员不需要主动去关心退出进程的事情。

（1）前台进程：指那些和用户正在进行的操作相关的进程。

（2）可视进程：尽管没有和用户交互，却可以影响用户所看到的内容。

（3）服务进程：凡是使用 startService()方法所启动的 service 对象，其所在的进程都称为服务进程。

（4）后台进程：不满足以上任何条件的进程，同时该进程中还包含一些不可见的 Activity，这些进程不影响正在和用户交互的 Activity。

（5）空进程：进程中不包含任何 component，包括 Activity、service、Receive 对象。

4. View 工作原理

1）消息处理过程

View 系统定义了从用户输入消息到消息处理的全过程，对于任何图形系统而言，该过程基本上都是相同的。通用图形系统的消息处理过程如图 3-5 所示。

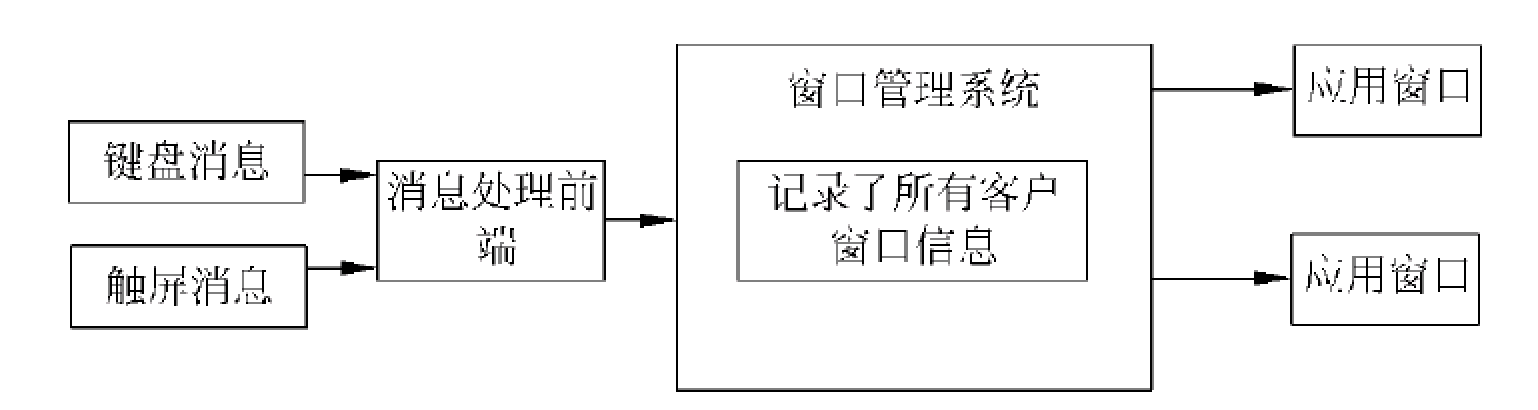


图 3-5 通用图形系统的消息处理过程

用户通过触摸屏或键盘等输入设备产生输入消息；该消息首先被消息处理前端转换为更明确的消息值；接下来，窗口管理系统根据所有窗口的状态判断用户正在与哪个窗口进行交互，然后把该消息发送给当前窗口；由于窗口是由 WmS 创建的，因此 WmS 知道所有客户的窗口信息，包括窗口的大小、位置等，当消息到达时，如果是按键消息则直接发送给当前窗口，而如果是触摸消息则 WmS 会根据消息的位置坐标去匹配所有的窗口，判断该坐标落到了哪一个窗口区域中，然后把该消息发送给相应的窗口；最后消息顺利地到达目标窗口。

View 系统获得消息后，会按照默认的逻辑来派发消息，主要就是把该消息派发给所有的子视图，以便相应的子视图能够获得消息并执行不同的任务。如果任务是一个纯后台任务，即该任务不会引起任何界面变换，那么 View 系统接下来仅仅按照默认逻辑继续派发下一个用户消息；而如果该任务会引起界面变换，那么 View 系统则要重新绘制界面。

2）重绘界面

重绘界面的过程大致分为以下 3 步：

(1) 计算该窗口中所有视图的大小，即测量出所有视图的实际尺寸大小。在 View 的内部逻辑中使用了一个内部变量保存相应的状态，当用户的某个操作导致改变了视图大小时会设置该变量，而 View 的内部逻辑会根据该变量决定是否需要重新测量。

(2) 为所有视图分配位置，即应该把视图放在屏幕的什么位置上。视图必须要给一个位置，该位置坐标相对于其父视图。不同视图的位置可以重叠，View 系统本身并不限定子视图的位置。Framework 中为了便于应用程序的设计，提供了许多不同的父视图，这些视图内部的子视图会按定义的方式自动获取位置。

(3) 把视图绘制到屏幕后，当系统获知了窗口中所有视图的大小和位置后，就可以完全确定屏幕应该显示哪些视图了。在绘制时，系统内部为每一个窗口创建了一个画布对象，并把这个画布对象传递给从根视图到所有子视图。

3）用户消息类型

用户消息类型分为以下 3 种：

(1) 按键消息。实现类是 android. view. KeyEvent，常用的 API 接口包括以下几项：

① getAction()：返回按键动作。只有两种动作，DOWN 或 UP。

② getKeyCode()：返回按键代码。这些代码是 Android 内部统一定义的，原始消息必须被转换成此代码才能被 Framework 处理。

③ getRepeat()：返回从按下后重复的次数。

(2) 触摸消息。实现类是 android. view. MotionEvent，常用的 API 接口包括以下几项：

① getAction()：获取消息动作。触摸消息动作的定义远远大于按键消息动作，因此触摸消息中必须包含是哪个点按下或者释放。

② getEventTime()和 getDownTime()：获取本次消息发生的时间和获取 DOWN 消息发生的时间。

③ getPressure()：获取用户触摸力量的大小，其值可以大于 1。

④ getSize()：近似反映了用户触摸面积的大小。

⑤ getX(int index)和 getY(int index)：返回指定触摸点对应的坐标。

(3) 轨迹消息。

4）按键消息派发过程

按键消息总体派发过程如下：

(1) 调用 mView. dispatchKeyEventPreIme()方法。

(2) 需要把该消息派发到输入法窗口，如果不存在则直接派发到真正的视图。

(3) 调用 deliverKeyEventToViewHierarchy()方法，将消息派发给真正的视图。该方法内部又可分为以下 3 个部分：

① 调用 checkForLeacingTouchModeAndConsume()方法，判断该信息是否会导致离开触摸模式并且消耗掉该消息，一般返回 false。

② 调用 mView. dispatchKeyEvent()方法将消息派发给根视图。

③ 如果应用程序中没有处理该消息，则默认会判断该消息是否会引起视图焦点的变换，如果会则进行焦点切换。

(4) 按键消息处理完毕,此时应该返回。总体调用过程:如果窗口存在输入法窗口,则先把按键消息交给输入法窗口处理,不过在处理之前程序员可以重载 preIme 函数以截获一些特别的按键消息;输入法如果没有消耗该消息,消息则进入 View 树进行处理,处理完所有消息后,如果 WmS 要求发送执行完毕回执,则调用 finishInputEvent()方法。

5) 触摸消息

触摸消息与按键消息的不同体现在以下几点:

(1) 触摸消息是消息获取模块直接派发给应用程序的。

(2) 触摸消息在处理时需要根据触摸坐标计算该消息应该派发给哪个 View,在按键消息处理中不存在该计算过程。

(3) 没有类似于系统按键的系统触摸键,应用程序可完全控制触摸行为。

(4) 子视图优先父视图消息处理,这与按键消息的处理完全相反。

触摸消息总体派发过程如下:

(1) 进行物理像素到逻辑像素的转换。

(2) 如果是 DOWN 消息,调用 ensureTouchMode(true)方法则进入触摸模式,与之相反的是按键模式。

(3) 将屏幕坐标转换到视图坐标。

(4) 调用 mView.dispatchTouchEvent()方法将消息派发给根视图,该方法内部会继而将消息派发到整个 View 树。

5. WmS 工作原理

WmS 是 Android 中图形用户接口的引擎,它管理着所有窗口,大致包括创建窗口、删除窗口以及将某个窗口设置为焦点窗口。焦点窗口是指当前正在和用户交互的窗口。

1) 窗口

在 WmS 中,窗口是由两部分内容构成的,一部分是描述该窗口的类 WindowState,另一部分是该窗口在屏幕上对应的界面 Surface。Surface 仅仅用于在屏幕上画少量界面,而负责将用户输入的触摸消息及按键消息派发到正在交互的窗口则与 Surface 没有关系。

2) 操作

在分析 WmS 内部逻辑中会进行 3 种常见的操作,具体操作可能会对应不同的函数名称,但是这些操作的语义是相同的。这 3 种常见的操作如下:

(1) Assign layer:为窗口分配层值。从用户的视角来看,层值越大的窗口越靠近用户。窗口之间的层叠正是按照层值进行的。

(2) Perform layout:计算窗口的大小。每个窗口对象都必须有一个大小,Perform layout 将根据状态栏大小、输入法窗口的状态、窗口动画状态计算该窗口的大小。

(3) Place surface:调整 Surface 对象的属性并重新将其显示到屏幕上。由于 assign layer 和 Perform surface 的执行结果影响的仅仅是 WindowState 类中的参数,而能够显示到屏幕上的窗口都包含一个 Surface 对象,因此只有将以上执行结果中的窗口层值、大小设置到 Surface 对象中,在屏幕上才能看出该窗口的变化。Place surface 的过程就是将这些值赋给 Surface 对象,并告诉 Surface Flinger 服务重新显示这些 Surface 对象。

3) 接口结构及交互过程

WmS 接口结构是指 WmS 功能模块与其他功能模块之间的交互接口,主要包括与 AmS 模块及应用程序客户端的接口,关系图如图 3-6 所示。

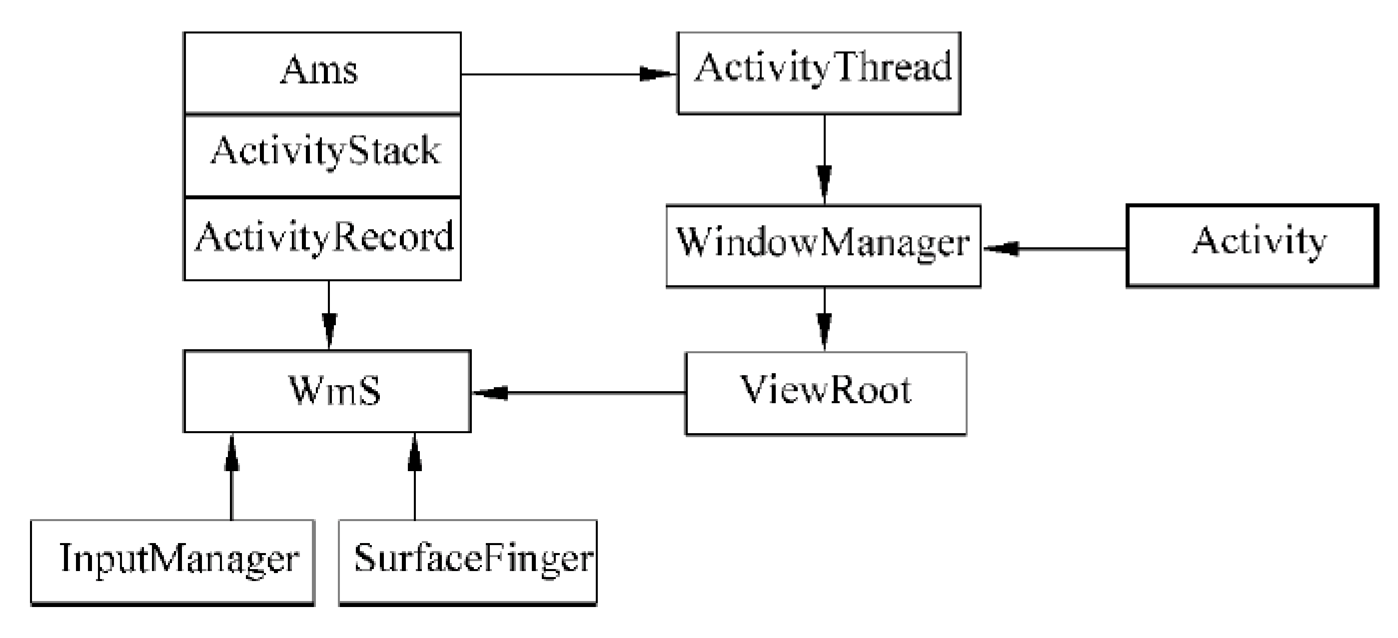


图 3-6　WmS 接口关系图

主要交互过程如下：

(1) 应用程序在 Activity 中添加、删除窗口。具体实现是通过调用 WindowManager 类的函数完成的，这会转而调用 ViewRoot 类的相关方法，然后通过调用到 WmS 中的相关方法完成添加、删除过程。

(2) 当 AmS 通知 ActivityThread 类会直接调用 WindowManager 类中的 removeView()方法删除窗口。

(3) AmS 中直接调用 WmS，告诉 WmS 一些其他信息。

6. 从输入设备中获取消息

1) 从输入设备中获取消息的过程

(1) 获取原始的用户消息，包括按键、触摸屏、鼠标、轨迹球等各种输入设备的消息。

(2) 对原始消息进行一定的加工，使之转换为程序可以理解的消息。

(3) 把转换后的消息发送到相应的用户窗口所在进程。如果消息获取线程和用户线程在同一个进程空间中，则传递消息比较简单；但对于多进程系统来说，消息获取线程和用户线程往往在不同的进程空间中，因此需要使用 IPC 机制把消息传递到用户窗口所在的线程中。

在 Android 2.2 的所有版本中，获取用户消息的方式基本上是相同的，其过程是在 WmS 中有一个子类 KeyQ，该类基于 KeyInputQueue 类，而该类内部则包含一个线程对象，该线程的任务是调用 native 方法从输入设备中读取用户消息，并把读取到的消息保存到一个 QueueEvent 队列中。

在 WmS 中有另外一个子类叫做 InputDispatcherThread，该类也是一个线程类，任务是从上面的 QueueEvent 队列中读取用户消息，并对这些消息进行一定的加工，然后判断应该把这个消息发送给哪个应用窗口。

在每个应用窗口对象 ViewRoot 中都包含一个 W 子类，该类是一个 Binder 类，InputDispatchThread 通过 IPC 方式调用 W 所提供的函数，从而把消息发送给对应的客户端窗口。

2) 与消息获取相关的源文件

(1) frameworks/base/services/java:com. android. service。主要是 WmS 端要执行的功能。这些文件中包括的 InputWindow 记录了客户窗口的信息，可以认为 InputWindow 是一个信息类。

(2) frameworks/base/core/java:ndroid. view. * 。这些文件主要是给客户端使用，其中 InputEvent 是所有输入消息的基类，是一个 abstract 类型，并且实现了 Parcelable 接口。即所有的输入消息都是可以跨进程传递的数据类。KeyEvent 和 MotionEvent 是

InputEvent 的两个实现，分别对应按键消息和触摸屏消息。

(3) Base/libs/ui。这些是 native 代码。其中 InputDispatcher 类就是进行消息派发的线程类，InputReader 是进行消息读取的线程类，InputTransport 是 native InputChannel 的实现类（这个文件名称或许叫做 InputChannel 才更合适），InputManager 是 native 的 InputManager 类。

3.4 Android 底层库和程序

3.4.1 本地实现底层的结构

Android 的本地实现底层具有基本的库和程序，这些库和程序是 Android 基本系统运行的基础。主要包括以下内容：

(1) 标准 C/C++库 bionic。

(2) C 语言底层库 libcutils。

(3) Init 进程。

(4) Shell 工具。

(5) C++工具库 libutils。

3.4.2 增加本地程序和库的方法

可以在 Android 中增加本地的程序和库，这些程序和库与它们所在的路径没有关系，只与它们的 Android. mk 文件有关系。Android. mk 文件具有统一的写法，主要包含了一些系统公共的宏。其选项参考以下文件：build/core/config. mk。默认的值在以下文件中定义：build/core/base_rules. mk。

在一个 Android. mk 文件中也可以生成多个可执行程序、动态库或者静态库。

1. 编译程序

1) 可执行程序的 Android. mk 文件

文件代码如下：

```
# Test Exe
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES := \
main.c
LOCAL_MODULE := test_exe
#LOCAL_C_INCLUDES :=
#LOCAL_STATIC_LIBRARIES :=
#LOCAL_SHARED_LIBRARIES :=
include $(BUILD_EXECUTABLE)
```

编译一个可执行程序，需要在 LOCAL_SRC_FILES 中加入源文件路径（相对于当前的目录），在 LOCAL_C_INCLUDES 中加入所需要包含的头文件路径；在 LOCAL_STATIC_LIBRARIES 中加入所需要链接的静态库（*.a）的名称；在 LOCAL_SHARED_LIBRARIES 中加入所需要链接的动态库（*.so）的名称。LOCAL_MODULE 表示模块最

终的名称。最后使用 include $(BUILD_EXECUTABLE)语句表示以一个可执行程序的方式进行编译。在本例中,LOCAL_MODULE 被定义为 test_exe,因此最终生成可执行程序的名称是 test_exe。

2) 静态库(归档文件)的 Android.mk 文件

文件代码如下:

```
# Test Static lib
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES := \
helloworld.c
LOCAL_MODULE := libtest_static
#LOCAL_C_INCLUDES :=
#LOCAL_STATIC_LIBRARIES :=
#LOCAL_SHARED_LIBRARIES :=
include $(BUILD_STATIC_LIBRARY)
```

编译一个静态库,基本内容和编译可执行程序相似,区别在于使用 include $(BUILD_STATIC_LIBRARY)语句表示编译静态库。在本例中,LOCAL_MODULE 被定义为 libtest_static,因此最终生成静态库的名称是 libtest_static.a。

3) 动态库(共享库)的 Android.mk 文件

文件代码如下:

```
# Test shared lib
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES := \
helloworld.c
LOCAL_MODULE := libtest_shared
TARGET_PRELINK_MODULE := false
#LOCAL_C_INCLUDES :=
#LOCAL_STATIC_LIBRARIES :=
#LOCAL_SHARED_LIBRARIES :=
include $(BUILD_SHARED_LIBRARY)
```

编译一个动态库,基本内容和编译可执行程序、静态库相似,区别在于使用 include $(BUILD_SHARED_LIBRARY)语句表示动态库。在本例中,LOCAL_MODULE 被定义为 libtest_shared,因此最终生成的动态库的名称是 libtest_shared.so。

4) 所在文件夹

可执行程序、动态库和静态库生成的列表分别在以下文件夹中:

out/target/product/generic/obj/EXECUTABLE;

out/target/product/generic/obj/STATIC_LIBRARY;

out/target/product/generic/obj/SHARED_LIBRARY。

其目标的文件夹分别为:

XXX_intermediates;

XXX_shared_intermediates；

XXX_static_intermediates。

对于可执行程序和动态库，生成的 LINK 子目录中包含带有符号的库（没有经过 strip）。

5）编译目标机的模板

```
include $(BUILD_EXECUTABLE)            #可执行程序
include $(BUILD_SHARED_LIBRARY)        #动态库
include $(BUILD_STATIC_LIBRARY)        #静态库
```

6）编译主机的模板

```
include $(BUILD_HOST_EXECUTABLE)       #可执行程序
include $(BUILD_HOST_SHARED_LIBRARY)   #动态库
include $(BUILD_HOST_STATIC_LIBRARY)   #静态库
```

7）安装路径的问题

用 LOCAL_MODULE_PATH 和 LOCAL_UNSTRIPPED_PATH 命令。增加以下语句可以安装到不同的文件系统：

```
LOCAL_MODULE_PATH := $(TARGET_ROOT_OUT)
LOCAL_UNSTRIPPED_PATH := $(TARGET_ROOT_OUT_UNSTRIPPED)
```

8）文件系统的选择

（1）TARGET_ROOT_OUT：表示根文件系统，路径为 out/target/product/generic/root。

（2）TARGET_OUT：表示 system 文件系统，路径为 out/target/product/generic/system。

（3）TARGET_OUT_DATA：表示 data 文件系统，路径为 out/target/product/generic/data。

2. 安装程序

除了编译各种内容外，有时还需要向目标文件系统复制一些文件，例如配置脚本、资源文件、预置的程序和库等，也有时需要在目标文件系统中创建目录。

在 Android.mk 文件中进行目录创建和安装工作，代码如下：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
copy_from := \
A.txt \
B.txt
copy_to := $(addprefix $(TARGET_OUT)/txt/, $(copy_from))
$(copy_to) : PRIVATE_MODULE := txt
$(copy_to) : $(TARGET_OUT)/txt/% : $(LOCAL_PATH)/% | $(ACP)
$(transform-prebuilt-to-target)
ALL_PREBUILT = $(copy_to)
# create some directories
DIRS := $(addprefix $(TARGET_OUT)/, \
System/txt
```

```
$(DIRS):
@echo Directory: $@
@mkdir -p $@
```

以上代码完成以下两个主要功能：

(1) 创建路径：system/txt。

(2) 将当前路径下的 A.txt 和 B.txt 文件复制到 system/txt 目录中。

3.4.3 标准 C/C++ 库 bionic

bionic 提供 C/C++标准库的功能，它是一个专为嵌入式系统设计的轻量级标准库实现。bionic 的源码和头文件在 bionic/目录中。

相对传统的标准库(如 glibc)实现，bionic 的体积和内存占用更小。bionic 支持标准 C/C++库的绝大部分功能，支持数学库以及 NPTL 线程库。它还实现了自己的 Linker 以及 Loader，用于动态库的创建和加载。

bionic 加入了一些 Android 独有的功能，例如 log 的底层支持。另外它还实现了一套 property 系统，这是整个 Android 的全局变量的存储区域，bionic 使用共享内存的方式来实现维护 property 系统。

3.4.4 C 语言底层库 libcutils

C 语言底层库提供了 C 语言中最基本的工具功能。这是 Android 本地中最为基础的库，基本上 Android 中所有本地的库和程序都链接了这个库。

头文件的路径：system/core/include/cutils。

库的路径：system/core/libcutils。

编译的结果是：libcutils.so。

libcutils 中主要的头文件：

(1) threads.h：线程。

(2) sockets.h：Android 的套接字。

(3) properties.h：Android 的属性。

(4) log.h：log 信息。

(5) array.h：数组。

(6) ashmem.h：匿名共享内存。

(7) atomic.h：原子操作。

(8) mq.h：消息队列。

3.4.5 Init 进程

Android 启动后，系统执行的第一个进程是一个名为 init 的可执行程序。它提供的功能包括：设备管理；解析启动脚本 init.rc；执行启动脚本中的基本功能；执行启动脚本中的各种服务。

代码的路径：system/core/init。

编译的结果是一个可执行文件：init。

启动脚本的路径：system/core/rootdir/init.rc。

1. init 的可执行文件

init 的可执行文件是系统运行的第一个用户空间的程序，它以守护进程的方式运行。代码如下：

```
int main(int argc, char ** argv)
{
/* ...... */
umask(0);
/* 创建文件系统的基本目录 */
open_devnull_stdio();                /* 打开 3 个文件 */
log_init();                          /* 初始化 log */
parse_config_file("/init.rc");       /* 处理初始化脚本 */
/* 获取内核命令行参数 */
qemu_init();
import_kernel_cmdline(0);
/* 初始化驱动设备,创建文件系统节点 */
device_fd = device_init();
/* 属性相关处理和启动 logo */
/* 初始化 struct pollfd ufds[4]; */
for(;;) {
/* 进入循环,处理 ufds[4]的事件 */
nr = poll(ufds, fd_count, timeout);
if (nr <= 0)
continue;
/* ...... */
}
return 0;
}
```

2. init.rc 文件

init.rc 是在 init 启动后被执行的启动脚本，其语法包含了 Actions、Triggers、Services、Options、Commands、Properties 等。代码如下：

```
on init
        export PATH /sbin:/system/sbin:/system/bin:/system/xbin
        mkdir /system
on property:ro.kernel.qemu=1
        start adbd
service vold /system/bin/vold
        socket vold stream 0660 root mount
```

使用方法参考：system/core/init/readme.txt。

关键字参考：system/core/init/keyword.h。

3.4.6 Shell 工具

Android 系统启动后提供了基本 Shell 界面供开发调试使用。需要启动一个名为 console 的服务，实际上执行的程序代码路径为/system/bin/sh。

sh 代码的路径：system/core/sh。

toolbox 代码的路径：system/core/toolbox。

生成文件/system/bin/toolbox,然后目标文件系统将/system/bin/中的相关符号连接到生成的文件/system/bin/toolbox 上。

3.4.7 C++工具库 libutils

libutils 是 Android 的底层库,这个库以 C++实现,它提供的 API 也是 C++的。Android 层次的 C 语言程序和库,大都基于 libutils 开发。libutils 工具库如图 3-7 所示。libutils 工具库基础类部分如图 3-8 所示。

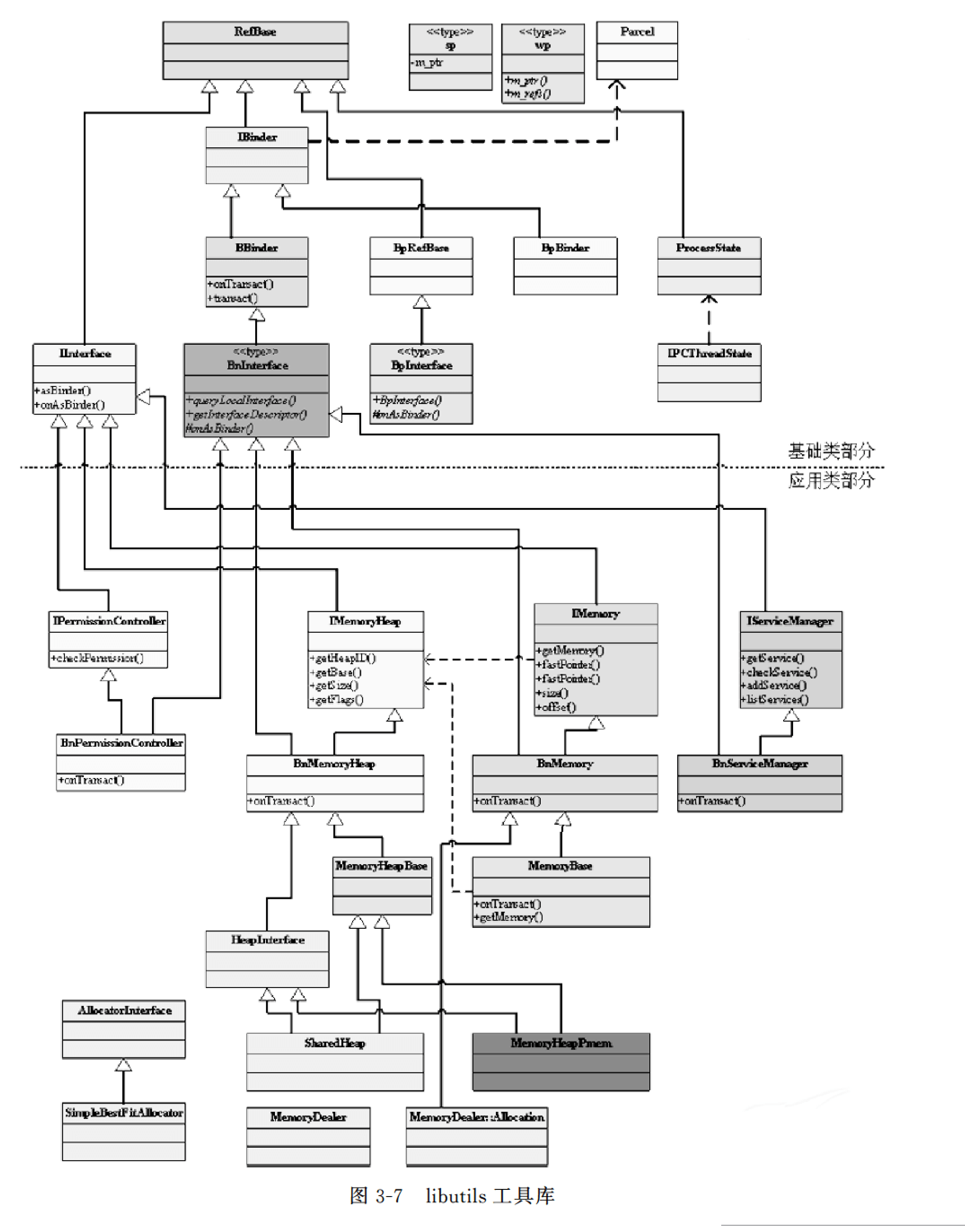


图 3-7 libutils 工具库

1. 基本信息

头文件的路径：frameworks/base/include/utils。

库的路径：frameworks/base/libs/utils。

编译的结果是：libutils. so。

2. libutils 中主要的头文件

(1) Errors. h：定义宏表示错误代码。

(2) Endian. h：定义表示大小端的宏。

(3) misc. h：几个字符串和文件相关的功能函数。

(4) TextOutput. h：定义文本输出的基类 TextOutput。

(5) BufferedTextOutput. h：BufferedTextOutput 类，它是一个 TextOutput 的实现。

(6) Pipe. h：定义管道类 Pipe。

(7) Buffer. h：定义内存缓冲区域的 Buffer 类。

(8) List. h：定义链表的模板类。

(9) SharedBuffer. h：定义 SharedBuffer 类表示共享内存。

(10) String16. h：定义表示双字节字符串的类 String16。

(11) String8. h：定义表示单字节字符串的类 String8，并包含了从 String16 类转换功能。

(12) VectorImpl. h：定义表示向量的类 VectorImpl。

(13) Vector. h：定义继承 VectorImpl 的类模板 Vector。

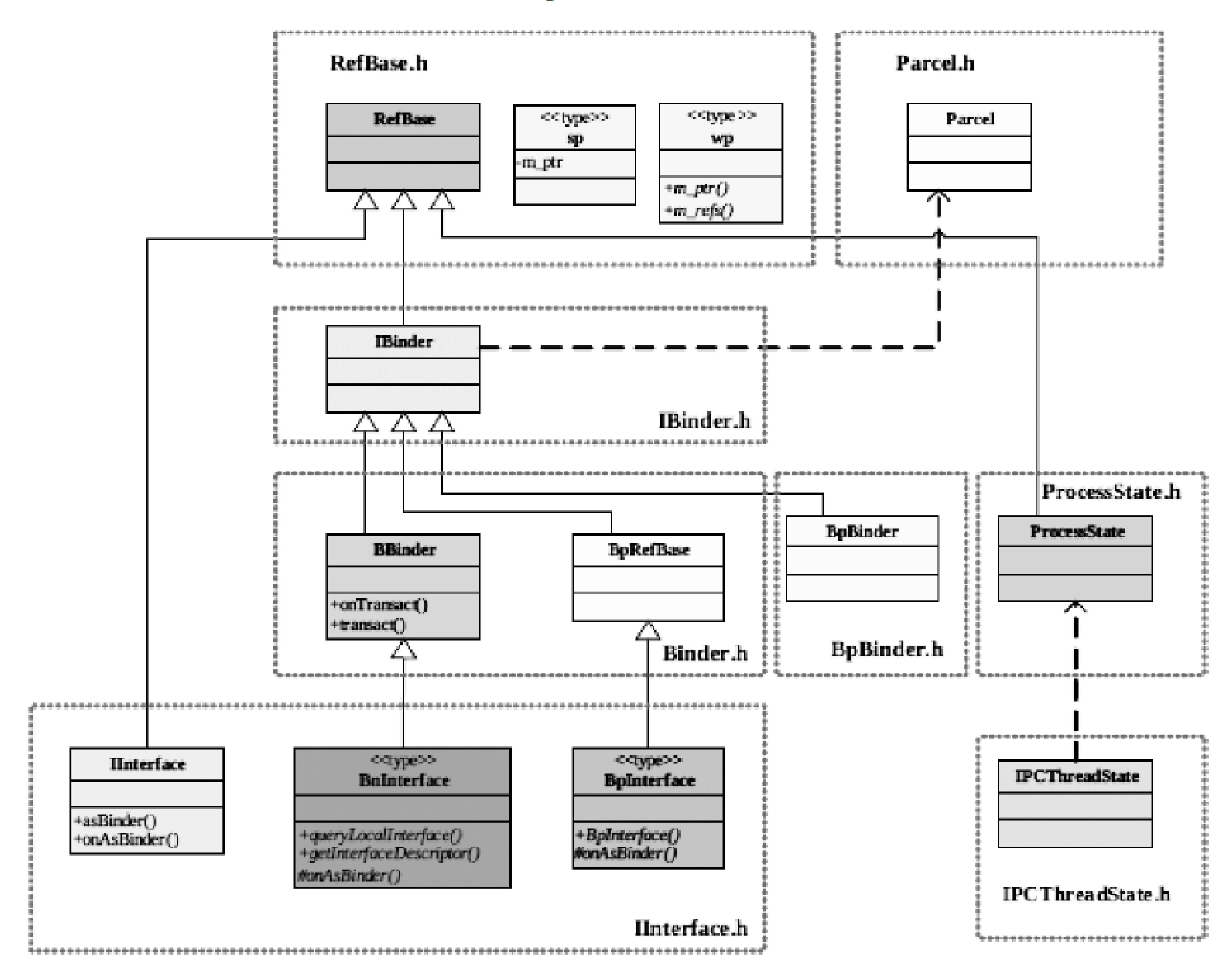


图 3-8　libutils 工具库基础类部分

（14）SortedVector.h：定义排序向量的模板 SortedVector。

（15）KeyedVector.h：定义使用关键字的向量模板 KeyedVector。

（16）threads.h：定义线程相关的类，包括线程 Thread、互斥量 Mutex、条件变量 Condition、读写锁 ReadWriteLock 等。

（17）socket.h：定义套结字相关的类 Socket。

（18）Timers.h：定义时间相关的函数和定时器类 DurationTimer。

（19）ZipEntry.h、ZipFileCRO.h、ZipFile.h、ZipFileRO.h、ZipUtils.h：与 zip 功能相关的类。

3．Binder

Binder 用于进程间的通信(IPC)的实现基础是运行于 Kernel 空间的 Binder 驱动。其示意图如图 3-9 所示。主要的头文件如下：

（1）RefBase.h:引用计数，定义类 RefBase。

（2）Parcel.h:为在 IPC 中传输的数据定义容器，定义类 Parcel。

（3）IBinder.h：Binder 对象的抽象接口，定义类 IBinder。

（4）Binder.h：Binder 对象的基本功能，定义类 Binder 和 BpRefBase。

（5）BpBinder.h：BpBinder 的功能，定义类 BpBinder。

（6）IInterface.h：为抽象经过 Binder 的接口定义通用类，定义类 IInterface、类模板 BnInterface、类模板 BpInterface。

（7）ProcessState.h：表示进程状态的类，定义类 ProcessState。

（8）IPCThreadState.h：表示 IPC 线程的状态，定义类 IPCThreadState。

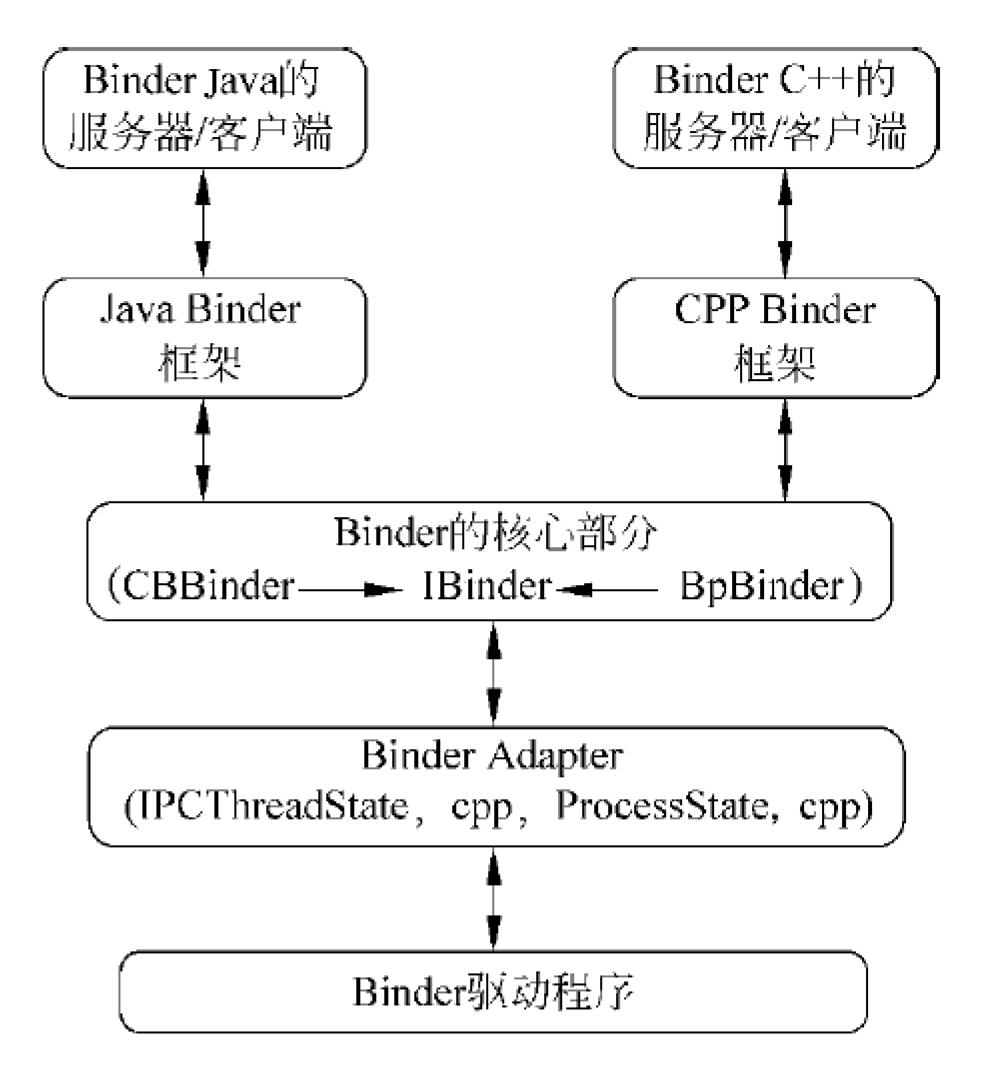


图 3-9　Binder 用于进程间的通信示意图

参考 PermissionController 的实现，这个在

```
IPermissionController .h
IPermissionController.cpp
```

进程间的通信示意图如图 3-10 所示。

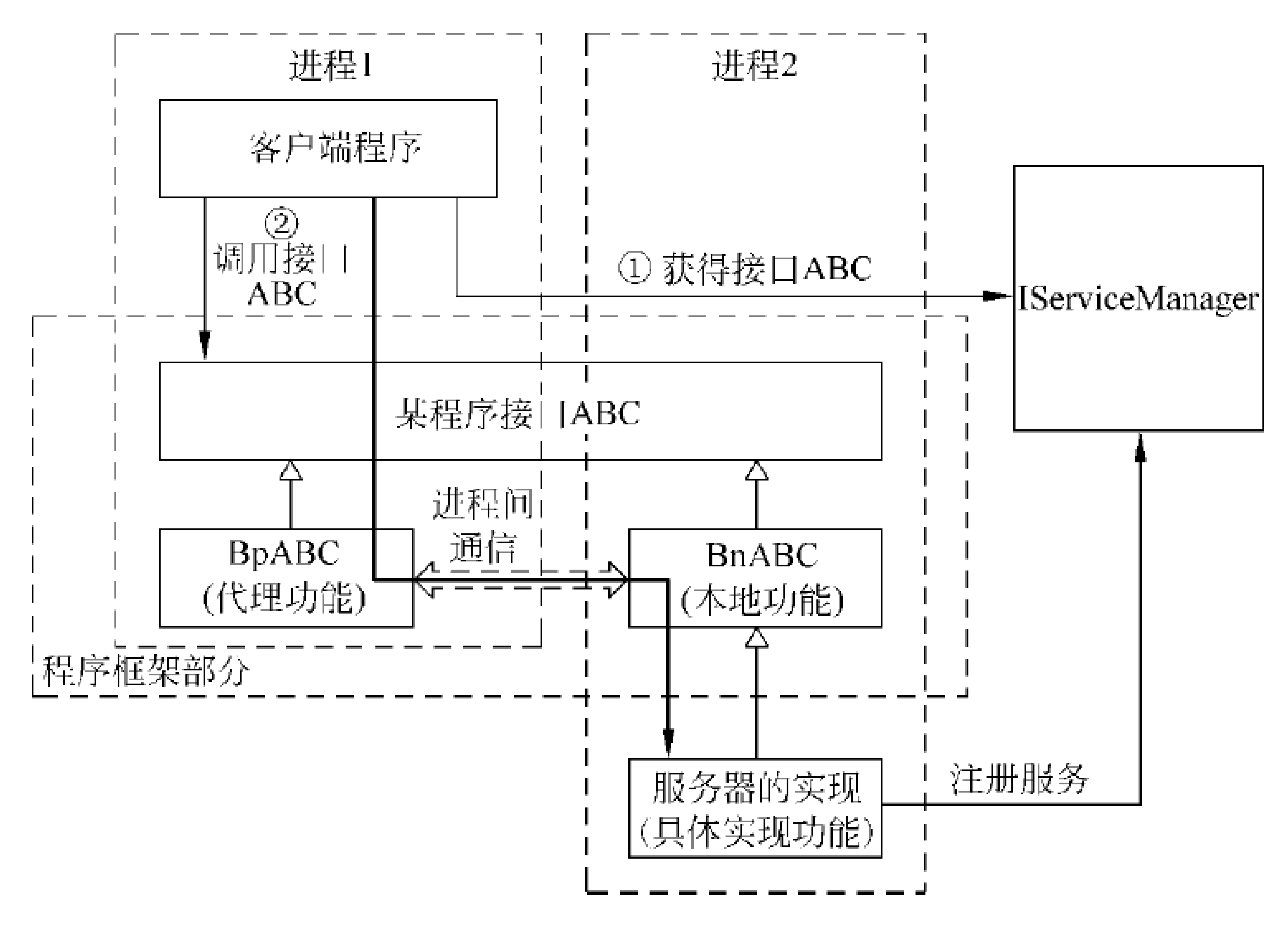


图 3-10　进程间的通信

CPP Binder 框架内定义接口 IPermissionController，并且实现了 BpPermisionController 类。

IPermissionController 类的使用方式是：

① 实现类继承 BnPermissionController；

② 调用者调用类 IPermissionController。

IPermissionController. h 的代码如下：

```
class IPermissionController : public IInterface
{
public:
DECLARE_META_INTERFACE(PermissionController);
virtual bool checkPermission(const String16& permission,
int32_t pid, int32_t uid) = 0;
enum {
CHECK_PERMISSION_TRANSACTION = IBinder::FIRST_CALL_TRANSACTION
};
};
class BnPermissionController : public BnInterface< IPermissionController >
{
public:
virtual status_t onTransact(uint32_t code,
const Parcel& data,
Parcel * reply,
uint32_t flags = 0);
};
IPermissionController .cpp(1)
```

```
class BpPermissionController : public BpInterface<IPermissionController>
{
public:
BpPermissionController(const sp<IBinder>& impl)
: BpInterface<IPermissionController>(impl) { }
virtual bool checkPermission(const String16& permission,
int32_t pid, int32_t uid)
{
Parcel data, reply;
data.writeInterfaceToken(IPermissionController::
getInterfaceDescriptor());
data.writeString16(permission);
data.writeInt32(pid);
data.writeInt32(uid);
remote()->transact (CHECK_PERMISSION_TRANSACTION, data, &reply);
if (reply.readInt32() != 0) return 0;
return reply.readInt32() != 0;
}
};
IMPLEMENT_META_INTERFACE(PermissionController,
"android.os.IPermissionController");
IPermissionController .cpp(2)
status_t BnPermissionController::onTransact(
uint32_t code, const Parcel& data, Parcel * reply, uint32_t flags)
{
switch(code) {
case CHECK_PERMISSION_TRANSACTION: {
CHECK_INTERFACE(IPermissionController, data, reply);
String16 permission = data.readString16();
int32_t pid = data.readInt32();
int32_t uid = data.readInt32();
bool res = checkPermission(permission, pid, uid);
reply->writeInt32(0);
reply->writeInt32(res ? 1 : 0);
return NO_ERROR;
} break;
default:
return BBinder::onTransact(code, data, reply, flags);
}
}
IPermissionController .cpp(2)
status_t BnPermissionController::onTransact(
uint32_t code, const Parcel& data, Parcel * reply, uint32_t flags)
{
switch(code) {
case CHECK_PERMISSION_TRANSACTION: {
```

```
CHECK_INTERFACE(IPermissionController, data, reply);
String16 permission = data.readString16();
int32_t pid = data.readInt32();
int32_t uid = data.readInt32();
bool res = checkPermission(permission, pid, uid);
reply->writeInt32(0);
reply->writeInt32(res ? 1 : 0);
return NO_ERROR;
} break;
default:
return BBinder::onTransact(code, data, reply, flags);
}
}
```

需要说明的是，Android 中几个重要系统进程为/init、/system/bin/servicemanager、/system/bin/mediaserver、system_server、Zygote，前面提到 init 通过解析 init.rc 启动对应的服务程序，servicemanager、Zygote 和 mediaserver 都是通过这种方式启动的，system_server 则是通过 Zygote 孵化出来的。这几个进程是 Android 系统运行的基础。

3.5 Android 的进程间通信机制 Binder

3.5.1 Binder 的提出

在 Android 系统中，每个应用程序都是由一些 Activity 和 Service 组成的。一般情况下，Service 运行在独立的进程中，而 Activity 有可能运行在同一个进程中，也有可能运行在不同的进程中。不在同一个进程中的 Activity 或者 Service 是通过 Binder 进程间通信机制进行通信的。

Android 系统是基于 Linux 内核的，而 Linux 内核继承和兼容了丰富的 UNIX 系统进程间通信(IPC)机制。有传统的管道(Pipe)、信号(Signal)和跟踪(Trace)，这 3 种通信手段只能用于父进程与子进程之间，或者兄弟进程之间；后来又增加了命名管道(Named Pipe)，使得进程间通信不再局限于父子进程或者兄弟进程之间；为了更好地支持商业应用中的事务处理，在 AT&T 的 UNIX 系统 V 中又增加了 3 种称为 System V IPC 的进程间通信机制，分别是报文队列(Message)、共享内存(Share Memory)和信号量(Semaphore)；后来 BSD UNIX 对 System V IPC 机制进行了重要扩充，提供了一种称为插口(Socket)的进程间通信机制。但是，Android 系统没有采用上述提到的各种进程间通信机制，而是采用 Binder 机制。

Binder 是一种进程间通信机制，它是一种类似于 COM 和 CORBA 的分布式组件结构，实际上提供远程过程调用(RPC)功能。Android 系统的 Binder 机制由 4 个组件组成，分别是 Client、Server、Service Manager 和 Binder 驱动程序，其中 Client、Server 和 Service Manager 运行在用户空间，Binder 驱动程序运行在内核空间。Binder 就是一种把这 4 个组件粘合在一起的黏结剂了，其中核心组件便是 Binder 驱动程序，Service Manager 提供辅助管理功能，Client 和 Server 正是在 Binder 驱动和 Service Manager 提供的基础设施上进行

Client-Server 之间的通信。Service Manager 和 Binder 驱动已经在 Android 平台中实现好，开发者只要按照规范实现自己的 Client 和 Server 组件就可以了。

3.5.2 Binder 概述

进程之间是拥有独立进程空间的，进程之间的数据是不能互相访问的，当进程间调用就要使用 Binder 机制。Binder 机制的核心是在客户端创建一个代理，在服务端创建一个存根，通过代理和存根之间的调用来完成进程间的数据交换。

1. 3 个程序名

(1) Android OS 的整个服务的管理程序：

```
<!--[if !supportLists]-->  <!--[endif]-->ServiceManager
```

(2) 在程序里边注册了提供媒体播放的服务程序 MediaPlayerService：

```
<!--[if !supportLists]-->  <!--[endif]-->MediaService
```

(3) 与 MediaPlayerService 交互的客户端程序：

```
<!--[if !supportLists]-->  <!--[endif]-->MediaPlayerClient
```

下面以 MediaService 为例来分析 Binder 的使用。

2. MediaService 应用程序

MediaService 是一个应用程序，本质上还是一个完整的 Linux 操作系统。MediaService 的源码文件在 framework\base\Media\MediaServer\Main_mediaserver.cpp 中，代码如下：

```
int main(int argc, char ** argv)
{
//获得一个 ProcessState 实例
sp<ProcessState> proc(ProcessState::self());
//得到一个 ServiceManager 对象
    sp<IServiceManager> sm = defaultServiceManager();
    MediaPlayerService::instantiate();          //初始化 MediaPlayerService 服务
    ProcessState::self()->startThreadPool();    //启动 Process 的线程池
    IPCThreadState::self()->joinThreadPool();   //将自己加入到刚才的线程池
}
```

1) 分析 MediaPlayerService

第一个调用的函数是 ProcessState::self()，然后赋值给了 proc 变量。程序运行完，proc 会自动删除内部的内容，所以就自动释放了先前分配的资源。

2) ProcessState

(1) ProcessState 位置在 framework\base\libs\binder\ProcessState.cpp 中，代码如下：

```
sp<ProcessState> ProcessState::self()
{
    if (gProcess != NULL) return gProcess;
    AutoMutex _l(gProcessMutex); --->锁保护
    if (gProcess == NULL) gProcess = new ProcessState; --->创建一个 ProcessState 对象
return gProcess; --->这里返回的是指针,但是函数返回的是 sp<xxx>,所以把 sp<xxx>看成
XXX*是可以的
}
```

（2）ProcessState 构造函数代码如下：

```
ProcessState::ProcessState()
    : mDriverFD(open_driver()) ----->调用函数
    , mVMStart(MAP_FAILED)//映射内存的起始地址
    , mManagesContexts(false)
    , mBinderContextCheckFunc(NULL)
    , mBinderContextUserData(NULL)
    , mThreadPoolStarted(false)
    , mThreadPoolSeq(1)
{
if (mDriverFD >= 0) {
//BIDNER_VM_SIZE 定义为(1*1024*1024) - (4096 *2) 1M-8K
        mVMStart = mmap(0, BINDER_VM_SIZE, PROT_READ, MAP_PRIVATE | MAP_NORESERVE,
mDriverFD, 0);//将 fd 映射为内存
    }
    ...
}
```

3）defaultServiceManager

（1）defaultServiceManager 位于 framework\base\libs\binder\IServiceManager.cpp 中，代码如下：

```
sp<IServiceManager> defaultServiceManager()
{
    if (gDefaultServiceManager != NULL) return gDefaultServiceManager;
    //又是一个单例,设计模式中叫 singleton.
    {
        AutoMutex _l(gDefaultServiceManagerLock);
        if (gDefaultServiceManager == NULL) {
    //真正的 gDefaultServiceManager 是在这里创建的
            gDefaultServiceManager = interface_cast<IServiceManager>(
                ProcessState::self()->getContextObject(NULL));//ProcessState::self,肯定
返回的是刚才创建的 gProcess,然后调用它的 getContextObject,注意传进去的是 NULL,即 0
        }
    }
   return gDefaultServiceManager;
}
```

（2）回到 ProcessState 类：

```
sp<IBinder> ProcessState::getContextObject(const sp<IBinder>& caller)
{
if (supportsProcesses()) {//该函数根据打开设备是否成功来判断是否支持 process,
//运行的时候一般都选择此步骤
        return getStrongProxyForHandle(0);//注意,这里传入 0
    }
}
```

（3）进入 getStrongProxyForHandle：

```
sp<IBinder> ProcessState::getStrongProxyForHandle(int32_t handle)
{
    sp<IBinder> result;
    AutoMutex _l(mLock);
handle_entry* e = lookupHandleLocked(handle);//从数组中查找对应索引的资源,内部会返回一
个 handle_entry
```

（4）下面是 handle_entry 的结构：

```
/*
    struct handle_entry {
                IBinder* binder; --->Binder
                RefBase::weakref_type* refs;
            };
*/
    if (e != NULL) {
        IBinder* b = e->binder; -->第一次进来为空
        if (b == NULL || !e->refs->attemptIncWeak(this)) {
            b = new BpBinder(handle); --->创建了一个新的 BpBinder
            e->binder = b;
            result = b;
        }....
    }
    return result; //返回刚才创建的 BpBinder
}
```

3.5.3 使用 Binder 进行进程间通信

案例：Activity 调用 Service，并且这两个组件分别在不同的进程内。

1. 编写 MyBinder 类，作为桥梁

```
public class MyBinder extends Binder {
    private Context cx;
    public MyBinder(Context cx) {
        this.cx = cx;
    }
    ...
```

```
//onTransact 函数是服务端收到客户端请求后对请求的处理函数:
protected boolean onTransact(int code, Parcel data, Parcel reply, int flags)
    throws RemoteException {
    try {
        reply.writeString("message receiver");
        Log.e ("==========Binder", "ontransact" + data.readString());
     } catch (Exception e) {
         e.printStackTrace();
     }
     return true;
}
```

2. 编写 Service 类作为服务端

```
public class MyService extends Service {
      private IBinder mBinder = null;
   public void onCreate() {
        Log.e ("==========MyService", "onCreate");
        mBinder = new MyBinder(getApplicationContext());
    }
    public IBinder onBind(Intent arg0) {
           Log.e ("==========MyService", "onBind");
           return mBinder;
      }
}
```

3. 编写配置文件

```
<service android:name=".MyService"   android:process=":remote">
                  <intent-filter>
                        <action android:name="zt.test2.Service" />
                  </intent-filter>
 </service>
```

4. 编写客户端 Activity,向 Service 发请求

```
public class ActivityServiceTest extends Activity {
    private IBinder ib = null;
    private String replystr;
    public void onCreate(Bundle savedInstanceState) {
             …
       bindService(new  Intent("zt.test2.Service"),mConnection,Context.BIND_AUTO_CREATE);
//绑定远程 Service
           …
//通过 Binder 的 transact 方法发送请求
public void onClick(View v) {
      Log.e ("==========Activity", "transact");
       try {
            Parcel pc = Parcel.obtain();
```

```
                Parcel pc_reply = Parcel.obtain();
                pc.writeString("i love this game");
                ib.transact(1, pc, pc_reply, 0);
                replystr = pc_reply.readString();
            } catch (RemoteException e) {
              e.printStackTrace();
            }
        }
});
private ServiceConnection mConnection = new ServiceConnection(){
    public void onServiceConnected(ComponentName arg0, IBinder arg1) {
        ib = arg1;
    }
    public void onServiceDisconnected(ComponentName arg0) {
    }
};
```

3.5.4 使用 AIDL 进行调用

传统的 Binder 方式只有一个函数可供通信，使用 AIDL（Android Interface Definition Language，Android 接口描述语言）可以生成多个函数被调用。因此一般选择 AIDL 这种方式，比前一种方式更直观、方便。

1. 编写 ITestService.aidl 文件

```
interface ITestService{
     int testipc(in String name);     //要使用 in 来修饰传入的参数
}
     public class DemoService extends Service {
     private Context mContext;
     private ITestService.Stub binder = new Stub(){
     public int testipc(String name) throws RemoteException {
          Log.e("============== stub", "==== name:" + name);
           return 12;
          }
     };
     public IBinder onBind(Intent intent) {
          mContext = this;
          return binder;
     }
}
<service android:name = "DemoService" android:process = ":remote">
          <intent - filter>
     <!-- AIDL 完整路径名.必须指明,客户端能够通过 AIDL 类名查找到它的实现类 -->
          <action android:name = "zt.demo.ITestService" />
      </intent - filter>
</service>
```

2. 在 Activity 中声明接口的引用

```
public class ActivityIPCTest extends Activity {
…
    private ITestService proxy;     //声明接口引用
…
}
```

3. 编写 ServiceConnection 类

```
private ServiceConnection sconnection = new ServiceConnection(){
    public void onServiceConnected(ComponentName name, IBinder service) {
        proxy = ITestService.Stub.asInterface(service);
        System.out.println("---------- proxy connected");
    }
    public void onServiceDisconnected(ComponentName name) {
        proxy = null;
    }
}
```

4. 绑定并使用 Proxy 代理

```
Button bbind = (Button) this.findViewById(R.id.bbind);        bbind.setOnClickListener(new
OnClickListener(){
public void onClick(View v) {
    try {
        bindService(new Intent(ITestService.class.getName()), sconnection, Context.BIND_
AUTO_CREATE);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
});
```

第 4 章　Android 系统开发

本章简单介绍 Android 系统的开发过程，使读者初步掌握 Android 系统的开发步骤。通过本章学习，读者应该掌握以下内容：

(1) 源码的获得。

(2) 源码树结构分析。

(3) Android 源码简要分析。

(4) Android 移植。

4.1　源码获得

作为一个 Android 开发者，必要的时候阅读以下源码可以拓宽一下自己的视野和对 Android 的认知程度。

Google 的 Android 的源码管理仓库用的是 Git。Android 是一个开源手机终端系统，基于 Linux 内核。

(1) 下载一个 Git 客户端，可以使用 Git-1.7.0.2 版本。下载地址为 http://code.google.com/p/msysgit/。下载界面如图 4-1 所示。

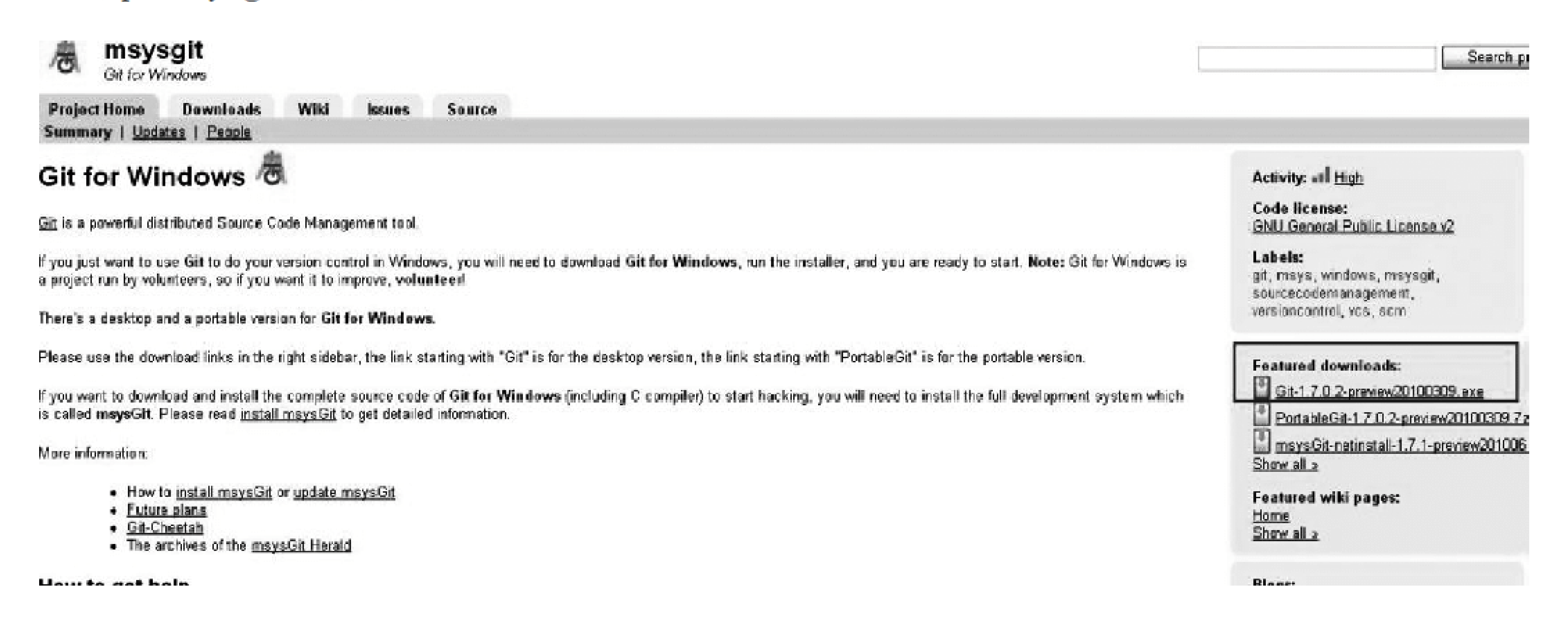


图 4-1　Git 客户端下载界面

(2) 在 Windows 下安装 Git 的客户端软件很方便，和普通软件一样。安装完成后，我们在计算机上建立一个文件夹，用来存放将要下载的源码。例如，在 D 盘上建立一个 android source 文件夹，如图 4-2 所示。



图 4-2　建立 android source 文件夹

（3）右击 android source 文件夹，选中 Git Bash 选项，会打开一个窗口命令，类似 CMD 命令窗口，如图 4-3 所示。命令窗口如下：

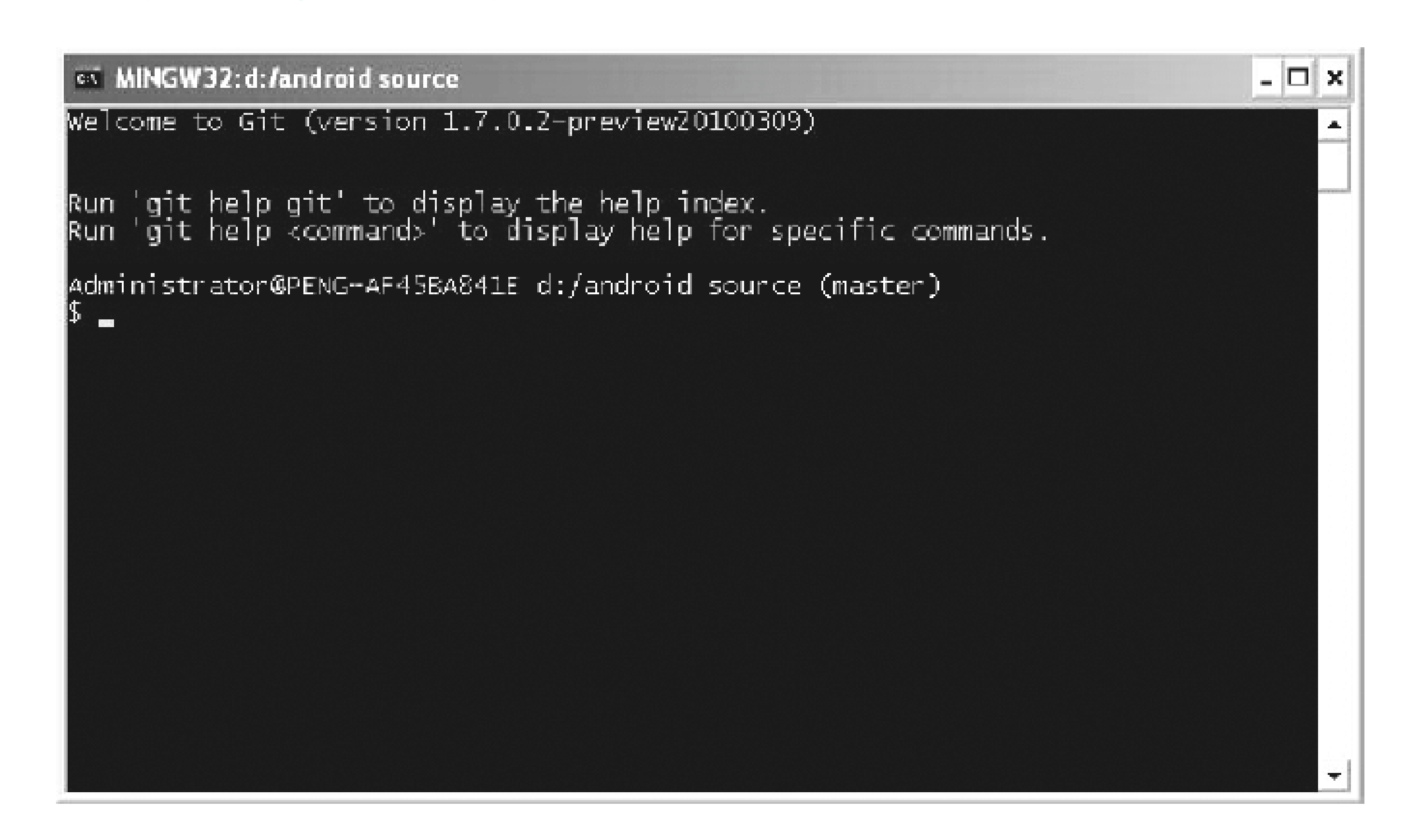


图 4-3　选中 Git Bash 后打开的命令窗口

（4）至此，源码获得方法基本完成一半了。接下来看看 Android 的源码地址，如图 4-4 所示。



Project	Description	Owner	Last Change	
device/common.git		Android Open Source...	4 months ago	summary | shortlog | log | tree
device/htc/common.git	Files specific to HTC devices...	Android Open Source...	3 weeks ago	summary | shortlog | log | tree
device/htc/dream-sapphire.git		Android Open Source...	4 days ago	summary | shortlog | log | tree
device/htc/dream.git	Files specific to HTC dream...	Android Open Source...	3 weeks ago	summary | shortlog | log | tree
device/htc/passion-common.git	Files specific to HTC passion...	Android Open Source...	3 weeks ago	summary | shortlog | log | tree

图 4-4　android 源码地址

源码地址：http://android.git.kernel.org/。

用浏览器打开此 URL 地址，可以看到，该地址首页告诉我们如何下载源码：git clone

git://android.git.kernel.org/＋工程模块的相对路径。例如，想要下载 platform/packages/apps/Launcher.git 这个模块，那么完整 URL 为：

```
git clone git://android.git.kernel.org/platform/packages/apps/Launcher.git
```

即在第三步打开的命令窗口中输入如上代码完成 url，按 Enter 键，即可将此模块源码下载到 android source 文件夹下，如图 4-5 所示。

```
MINGW32:d:/android source
Welcome to Git (version 1.7.0.2-preview20100309)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Administrator@PENG-AF45BA841E d:/android source (master)
$ git clone git://android.git.kernel.org/platform/packages/apps/launcher.git
Initialized empty Git repository in D:/android source/launcher/.git/
fatal: The remote end hung up unexpectedly

Administrator@PENG-AF45BA841E d:/android source (master)
$

Administrator@PENG-AF45BA841E d:/android source (master)
$ git clone git://android.kernel.org/platform/packages/apps/Launcher.git
Initialized empty Git repository in D:/android source/Launcher/.git/
remote: Counting objects: 3067, done.
remote: Compressing objects: 100% (850/850), done.
remote: Total 3067 (delta 1484), reused 3049 (delta 1470)
Receiving objects: 100% (3067/3067), 7.94 MiB | 16 KiB/s, done.
Resolving deltas: 100% (1484/1484), done.

Administrator@PENG-AF45BA841E d:/android source (master)
$
```

图 4-5　将模块源码下载到 android source 文件夹下

下载完成后，打开目标文件夹 android source，就会得到想要的源码，如图 4-6 所示。

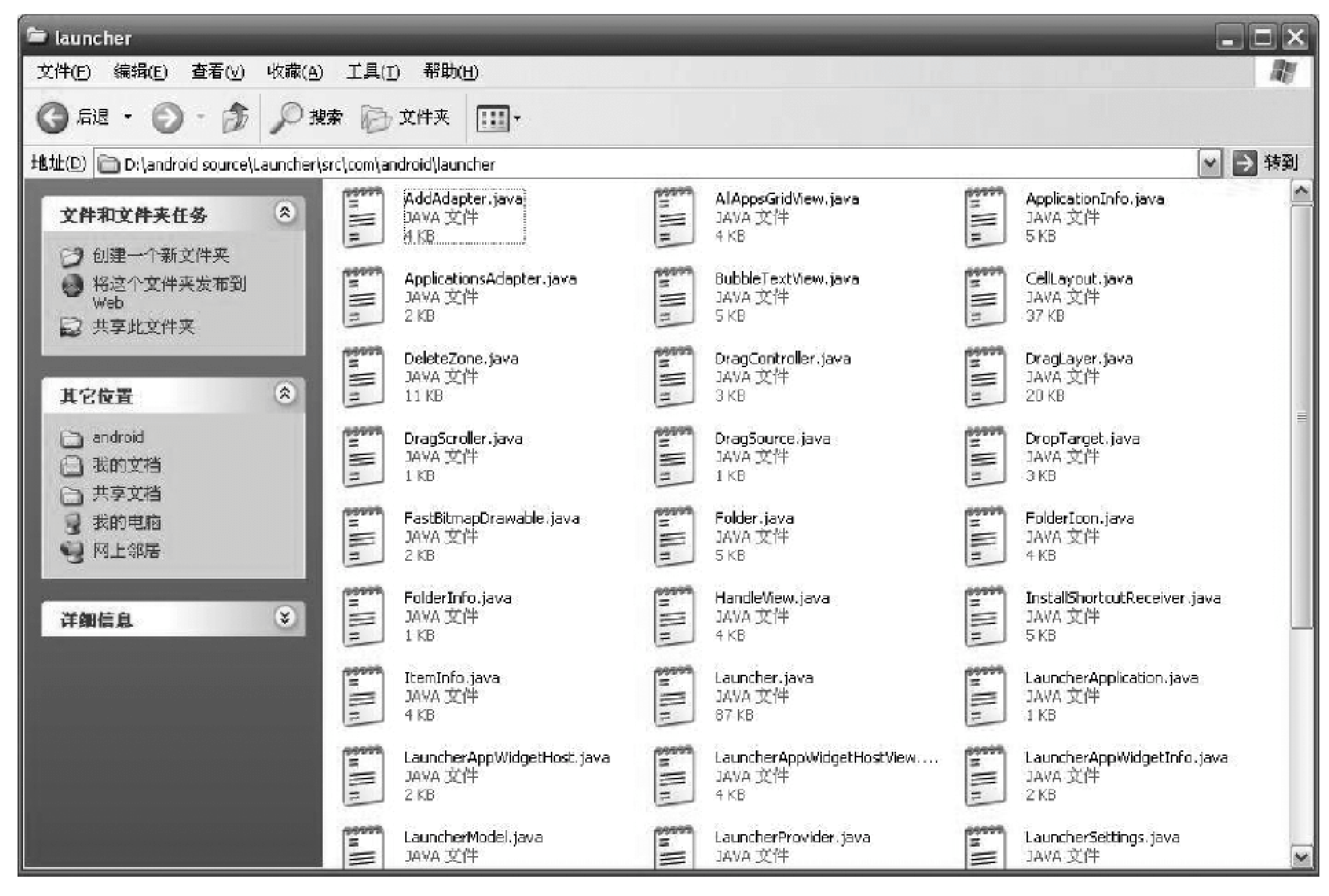


图 4-6　源码目标文件夹

至此，在 Windows 下获取 Android 源码的方法就结束了。

接下来看看在 Linux 下如何获取呢。

在这里使用的是 Ubuntu10.4，也可以使用 SuSE，RedHat。在 Ubuntu 10.4 上安装 Git 只要设定了正确的更新源，然后使用 apt-get 就可以了，有什么依赖问题，就让它自己解决吧。其中，curl 是一个利用 URL 语法在命令行下工作的文件传输工具。

在 Ubuntu Linux 中打开终端命令窗口，输入：sudo apt-get install git-core curl，如图 4-7 所示。

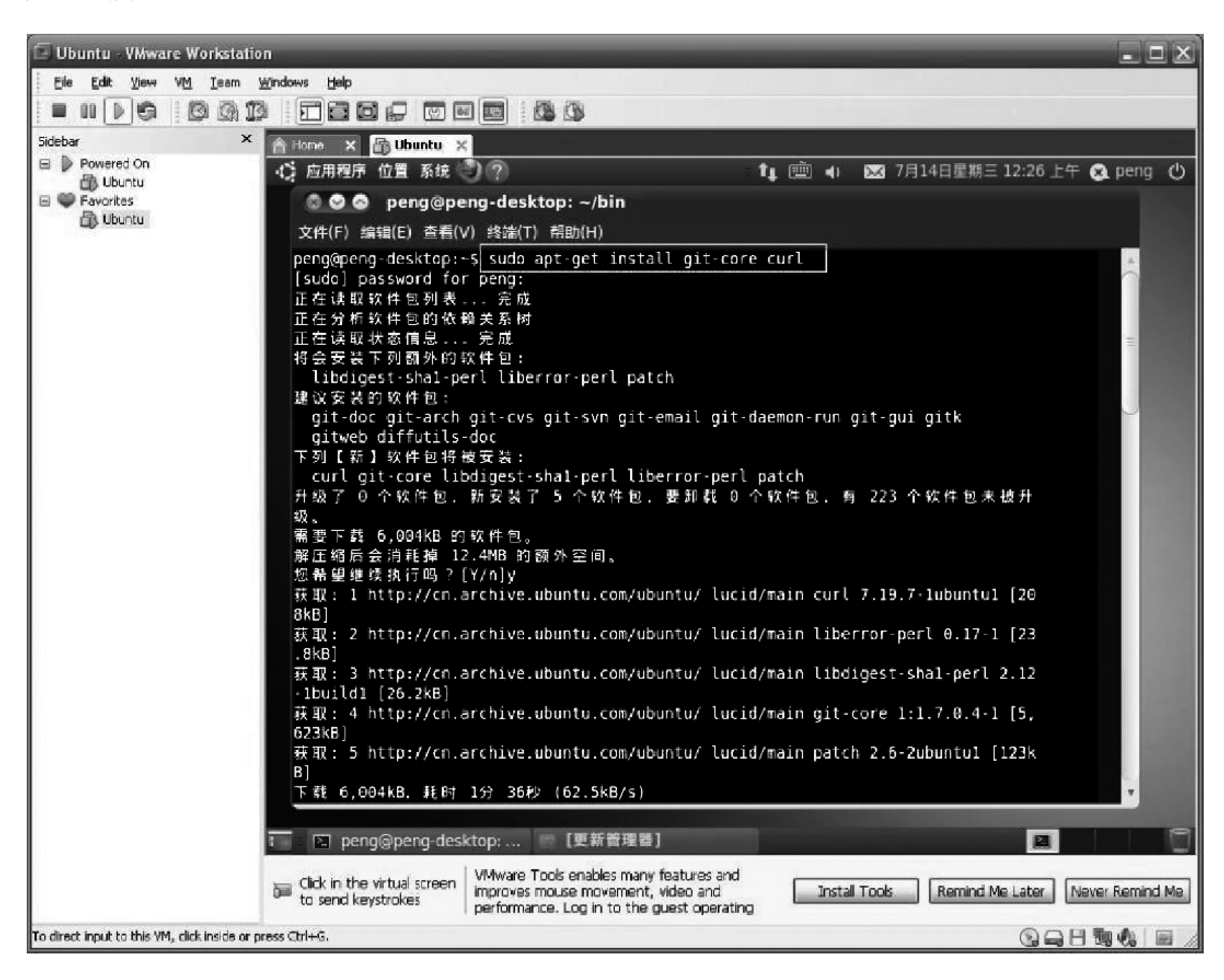


图 4-7 Ubuntu Linux 的终端命令窗口

安装完成之后，Git 会自动更新它需要的组件和依赖包。同时，读取一个目录，例如 android，更改权限“chmod 777/android”后，读写权限都附上。

输入的命令和 Windows 下的命令窗口一样，如果想获取 platform/packages/apps/Launcher.git 这个模块源码，输入如下命令：

```
git clone git://android.git.kernel.org/platform/packages/apps/Launcher.git
```

这些都是同一个道理。下载完成后如图 4-8 所示。

如果要全部下载下来，也是一样，获取 platform/manifest.git 即可。

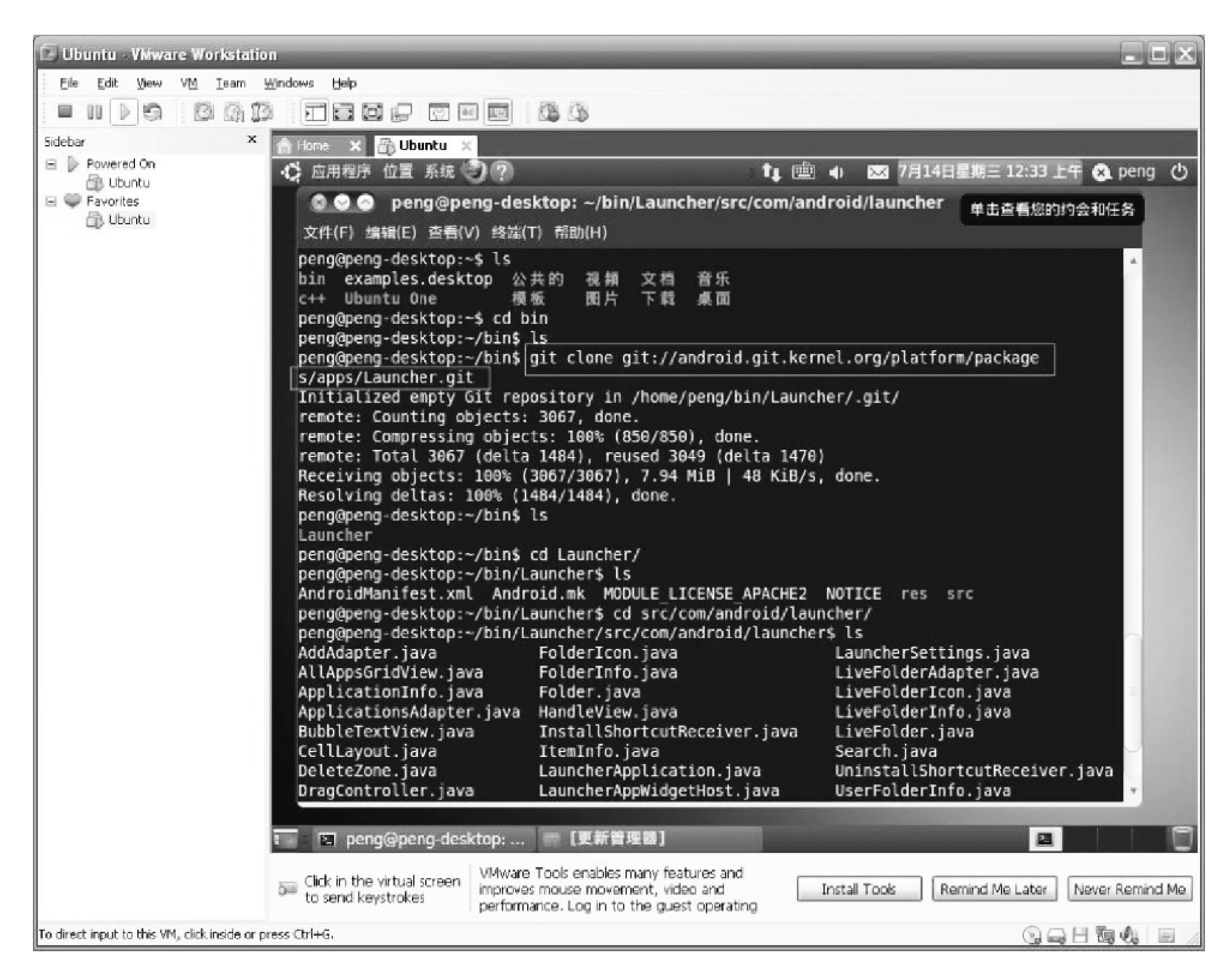


图 4-8　获取 platform/packages/apps/Launcher.git 模块源码图

4.2　源码结构分析

Google 提供的 Android 包含了原始 Android 的目标机代码、主机编译工具和仿真环境。代码包经过解压缩后，第一级别的目录和文件如下所示。其中：

1 |-- Makefile

2 |-- bionic(bionic C 库)

3 |-- bootable(启动引导相关代码)

4 |-- build(存放系统编译规则及 generic 等基础开发包配置)

5 |-- cts(Android 兼容性测试套件标准)

6 |-- dalvik(dalvik Java 虚拟机)

7 |-- development(应用程序开发相关)

8 |-- external(Android 使用的一些开源的模组)

9 |-- frameworks(核心框架——Java 及 C++语言)

10 |-- hardware(部分厂家开源的硬解适配层 HAL 代码)

11 |-- out(编译完成后的代码输出与此目录)

12 |-- packages(应用程序包)

13 |-- prebuilt(x86 和 arm 架构下预编译的一些资源)

```
|-- sdk(sdk 及模拟器)
|-- system(底层文件系统库、应用及组件——C 语言)
|-- vendor(厂商定制代码)
|--bionic 目录
|-- libc(C 库)
| |-- arch-arm(ARM 架构,包含系统调用汇编实现)
| |-- arch-x86(x86 架构,包含系统调用汇编实现)
| |-- bionic(由 C 实现的功能,架构无关)
| |-- docs(文档)
| |-- include(头文件)
| |-- inet(inet 相关)
| |-- kernel(Linux 内核中的一些头文件)
| |-- netbsd(nesbsd 系统相关)
| |-- private(一些私有的头文件)
| |-- stdio(stdio 实现)
| |-- stdlib(stdlib 实现)
| |-- string(string 函数实现)
| |-- tools(几个工具)
| |-- tzcode(时区相关代码)
| |-- unistd(unistd 实现)
| '-- zoneinfo(时区信息)
|-- libdl(libdl 实现,dl 是动态链接,提供访问动态链接库的功能)
|-- libm(libm 数学库的实现)
| |-- alpha(alpha 架构)
| |-- amd64(amd64 架构)
| |-- arm(arm 架构)
| |-- bsdsrc(bsd 的源码)
| |-- i386(i386 架构)
| |-- i387(i387 架构)
| |-- ia64(ia64 架构)
| |-- include(头文件)
| |-- man(数学函数,后缀名为.3,一些为 freeBSD 的库文件)
| |-- powerpc(powerpc 架构)
| |-- sparc64(sparc64 架构)
| '-- src(源代码)
|-- libstdc++(libstdc++C++实现库)
| |-- include(头文件)
| '-- src(源码)
|-- libthread_db(多线程程序的调试器库)
```

```
| '-- include(头文件)
'-- linker(动态链接器)
'-- arch(支持 arm 和 x86 两种架构)
bootable 目录
|-- bootloader(适合各种 bootloader 的通用代码)
| '-- legacy(估计不能直接使用,可以参考)
||-- arch_armv6(V6 架构,几个简单的汇编文件)
||-- arch_msm7k(高通 7k 处理器架构的几个基本驱动)
||-- include(通用头文件和高通 7k 架构头文件)
||-- libboot(启动库)
||-- libc(一些常用的 c 函数)
||-- nandwrite(nandwirte 函数实现)
| '-- usbloader(usbloader 实现)
|-- diskinstaller(android 镜像打包器,x86 可生产 iso)
'-- recovery(系统恢复相关)
|-- edify(升级脚本使用的 edify 脚本语言)
|-- etc(init.rc 恢复脚本)
|-- minui(一个简单的 UI)
|-- minzip(一个简单的压缩工具)
|-- mtdutils(mtd 工具)
|-- res(资源)
| '-- images(一些图片)
|-- tools(工具)
| '-- ota(OTA Over The Air Updates 升级工具)
'-- updater(升级器)
build 目录
|-- core(核心编译规则)
|-- history(历史记录)
|-- libs
| '-- host(主机端库,有 android "cp"功能替换)
|-- target(目标机编译对象)
||-- board(开发平台)
|||-- emulator(模拟器)
|||-- generic(通用)
|||-- idea6410(自己添加的)
|| '-- sim(最简单)
| '-- product(开发平台对应的编译规则)
| '-- security(密钥相关)
'-- tools(编译中主机使用的工具及脚本)
```

```
|-- acp(Android "acp" Command)
|-- apicheck(api 检查工具)
|-- applypatch(补丁工具)
|-- apriori(预链接工具)
|-- atree(tree 工具)
|-- bin2asm(bin 转换为 asm 工具)
|-- check_prereq(检查编译时间戳工具)
|-- dexpreopt(模拟器相关工具)
|-- droiddoc(java 语言,JDK5 有相关文档)
|-- fs_config(This program takes a list of files and directories)
|-- fs_get_stats(获取文件系统状态)
|-- iself(判断是否 ELF 格式)
|-- isprelinked(判断是否 prelinked)
|-- kcm(按键相关)
|-- lsd(List symbol dependencies)
|-- releasetools(生成镜像的工具及脚本)
|-- rgb2565(rgb 转换为 565)
|-- signapk(apk 签名工具)
|-- soslim(strip 工具)
'-- zipalign(zip archive alignment tool)
dalvik 目录(dalvik 虚拟机)
|-- dalvikvm(main.c 的目录)
|-- dexdump(dex 反汇编)
|-- dexlist(List all methods in all concrete classes in a dex file.)
|-- dexopt(预验证与优化)
|-- docs(文档)
|-- dvz(和 zygote 相关的一个命令)
|-- dx(dx 工具,将多个 java 转换为 dex)
|-- hit(Java 语言写成)
|-- libcore(核心库)
|-- libcore-disabled(禁用的库)
|-- libdex(dex 的库)
|-- libnativehelper(Support functions for Android's class libraries)
|-- tests(测试代码)
|-- tools(工具)
'-- vm(虚拟机实现)
development 目录(开发者需要的一些例程及工具)
|-- apps(一些核心应用程序)
||-- BluetoothDebug(蓝牙调试程序)
```

```
|   |-- CustomLocale(自定义区域设置)
|   |-- Development(开发)
|   |-- Fallback(和语言相关的一个程序)
|   |-- FontLab(字库)
|   |-- GestureBuilder(手势动作)
|   |-- NinePatchLab
|   |-- OBJViewer(OBJ 查看器)
|   |-- SdkSetup(SDK 安装器)
|   |-- SpareParts(高级设置)
|   |-- Term(远程登录)
|   '-- launchperf(装载前的预处理)
|-- build(编译脚本模板)
|-- cmds(有个 monkey 工具)
|-- data(配置数据)
|-- docs(文档)
|-- host(主机端 USB 驱动等)
|-- ide(集成开发环境)
|-- ndk(本地开发套件——C 语言开发套件)
|-- pdk(Plug Development Kit)
|-- samples(例程)
|   |-- AliasActivity
|   |-- ApiDemos(API 演示程序)
|   |-- BluetoothChat(蓝牙聊天)
|   |-- BrowserPlugin(浏览器插件)
|   |-- BusinessCard(商业卡)
|   |-- Compass(指南针)
|   |-- ContactManager(联系人管理器)
|   |-- CubeLiveWallpaper(动态壁纸的一个简单例程)
|   |-- FixedGridLayout(像是布局)
|   |-- GlobalTime(全球时间)
|   |-- HelloActivity(Hello)
|   |-- Home(Home)
|   |-- JetBoy(jetBoy 游戏)
|   |-- LunarLander(游戏)
|   |-- MailSync(邮件同步)
|   |-- MultiResolution(多分辨率)
|   |-- MySampleRss(RSS)
|   |-- NotePad(记事本)
|   |-- RSSReader(RSS 阅读器)
```

```
||-- SearchableDictionary(目录搜索)
||-- SimpleJNI(JNI 例程)
||-- SkeletonApp(空壳 APP)
||-- Snake(snake 程序)
||-- SoftKeyboard(软键盘)
||-- Wiktionary(维基)
|'-- WiktionarySimple(维基例程)
|-- scripts(脚本)
|-- sdk(sdk 配置)
|-- simulator(模拟器)
|-- testrunner(测试用)
'-- tools(一些工具)
external 目录
|-- aes(AES 加密)
|-- apache-http(网页服务器)
|-- astl(ASTL(Android STL)is a slimmed-down version of the regular C++STL.)
|-- bison(自动生成语法分析器,将无关文法转换成 C、C++)
|-- blktrace(blktrace is a block layer IO tracing mechanism)
|-- bluetooth(蓝牙相关、协议栈)
|-- bsdiff(diff 工具)
|-- bzip2(压缩工具)
|-- clearsilver(html 模板系统)
|-- dbus(低延时、低开销、高可用性的 IPC 机制)
|-- dhcpcd(DHCP 服务)
|-- dosfstools(DOS 文件系统工具)
|-- dropbear(SSH2 的 server)
|-- e2fsprogs(EXT2 文件系统工具)
|-- elfcopy(复制 ELF 的工具)
|-- elfutils(ELF 工具)
|-- embunit(Embedded Unit Project)
|-- emma(java 代码覆盖率统计工具)
|-- esd(Enlightened Sound Daemon,将多种音频流混合在一个设备上播放)
|-- expat(Expat is a stream-oriented XML parser.)
|-- fdlibm(FDLIBM(Freely Distributable LIBM))
|-- freetype(字体)
|-- fsck_msdos(dos 文件系统检查工具)
|-- gdata(google 的无线数据相关)
|-- genext2fs(genext2fs generates an ext2 filesystem as anormal(non-root)user)
|-- giflib(gif 库)
```

```
|-- googleclient(google 用户库)
|-- grub(This is GNU GRUB, the GRand Unified Bootloader. )
|-- gtest(Google C++ Testing Framework)
|-- icu4c(ICU(International Component for Unicode 在 C/C++下的版本)
|-- ipsec-tools(This package provides a way to use the native IPsec functionality )
|-- iptables(防火墙)
|-- jdiff(generate a report describing the difference between two public Java APIs. )
|-- jhead(jpeg 头部信息工具)
|-- jpeg(jpeg 库)
|-- junit(JUnit 是一个 Java 语言的单元测试框架)
|-- kernel-headers(内核的一些头文件)
|-- libffi(libffi is a foreign function interface library. )
|-- libpcap(网络数据包捕获函数)
|-- libpng(png 库)
|-- libxml2(xml 解析库)
|-- mtpd(一个命令)
|-- netcat(simple Unix utility which reads and writes dataacross network connections)
|-- netperf(网络性能测量工具)
|-- neven(看代码和 JNI 相关)
|-- opencore(多媒体框架)
|-- openssl(SSL 加密相关)
|-- openvpn(VPN 开源库)
|-- oprofile(OProfile 是 Linux 内核支持的一种性能分析机制)
|-- ping(ping 命令)
|-- ppp(pppd 拨号命令,没有 chat)
|-- proguard(Java class file shrinker, optimizer,obfuscator, and preverifier)
|-- protobuf(a flexible, efficient, automated mechanism for serializing structured data)
|-- qemu(arm 模拟器)
|-- safe-iop(functions for performing safe integer operations)
|-- skia(skia 图形引擎)
|-- sonivox(sole MIDI solution for Google Android Mobile Phone Platform)
|-- speex(Speex 编/解码 API 的使用(libspeex))
|-- sqlite(数据库)
|-- srec(Nuance 公司提供的开源连续非特定人语音识别)
|-- strace(trace 工具)
|-- svox(Embedded Text-to-Speech)
|-- tagsoup(TagSoup 是一个 Java 开发符合 SAX 的 HTML 解析器)
|-- tcpdump(抓 TCP 包的软件)
|-- tesseract(Tesseract Open Source OCR Engine. )
```

```
|-- tinyxml(TinyXml is a simple, small, C++XML parser)
|-- tremor(I stream and file decoder provides an embeddable,integer-only library)
|-- webkit(浏览器核心)
|-- wpa_supplicant(无线网卡管理)
|-- xmlwriter(XML 编辑工具)
|-- yaffs2(yaffs 文件系统)
'-- zlib(a general purpose data compression library)
frameworks 目录(核心框架——Java 及 C++语言)
|-- base(基本内容)
||-- api(都是 xml 文件,定义了 Java 的相关 api)
||-- awt(AWT 库)
||-- build(空的)
||-- camera(摄像头服务程序库)
||-- cmds(重要命令:am、app_proce 等)
||-- core(核心库)
||-- data(字体和声音等数据文件)
||-- docs(文档)
||-- graphics(图形相关)
||-- include(头文件)
||-- keystore(和数据签名证书相关)
||-- libs(库)
||-- location(地区库)
||-- media(媒体相关库)
||-- obex(蓝牙传输库)
||-- opengl(2D-3D 加速库)
||-- packages(设置、TTS、VPN 程序)
||-- sax(XML 解析器)
||-- services(各种服务程序)
||-- telephony(电话通信管理)
||-- test-runner(测试工具相关)
||-- tests(各种测试)
||-- tools(工具)
||-- vpn(VPN)
|'-- wifi(无线网络)
|-- opt(可选部分)
||-- com.google.android(有个 framework.jar)
||-- com.google.android.googlelogin(有个 client.jar)
|'-- emoji(standard message elements)
'-- policies(Product policies are operating system directions aimed at specific uses)
```

```
'-- base
|-- mid(MID 设备)
'-- phone(手机类设备一般用这个,与锁屏有关的代码)
hardware 目录(部分厂家开源的硬解适配层 HAL 代码)
|-- broadcom(博通公司)
| '-- wlan(无线网卡)
|-- libhardware(硬件库)
||-- include(头文件)
| '-- modules(Default(and possibly architecture dependents)HAL modules)
||-- gralloc(gralloc 显示相关)
| '-- overlay(Skeleton for the "overlay" HAL module.)
|-- libhardware_legacy(旧的硬件库)
||-- flashlight(背光)
|-- gps(GPS)
||-- include(头文件)
||-- mount(旧的挂载器)
||-- power(电源)
||-- qemu(模拟器)
|-- qemu_tracing(模拟器跟踪)
||-- tests(测试)
||-- uevent(uevent)
||-- vibrator(震动)
| '-- wifi(无线)
|-- msm7k(高通 7k 处理器开源抽象层)
||-- boot(启动)
||-- libaudio(声音库)
||-- libaudio-qsd8k(qsd8k 的声音相关库)
||-- libcamera(摄像头库)
||-- libcopybit(copybit 库)
||-- libgralloc(gralloc 库)
|-- libgralloc-qsd8k(qsd8k 的 gralloc 库)
||-- liblights(背光库)
| '-- librpc(RPC 库)
|-- ril(无线电抽象层)
||-- include(头文件)
|-- libril(库)
||-- reference-cdma-sms(cdma 短信参考)
||-- reference-ril(ril 参考)
| '-- rild(ril 后台服务程序)
```

```
'-- ti(ti 公司开源 HAL)
|-- omap3(omap3 处理器)
||-- dspbridge(DSP 桥)
||-- libopencorehw(opencore 硬件库)
|-- liboverlay(overlay 硬件库)
||-- libstagefrighthw(stagefright 硬件库)
|'-- omx(omx 组件)
'-- wlan(无线网卡)
Out 目录
packages 目录
|-- apps(应用程序库)
||-- AlarmClock(闹钟)
||-- Bluetooth(蓝牙)
||-- Browser(浏览器)
|-- Calculator(计算器)
||-- Calendar(日历)
||-- Camera(相机)
||-- CertInstaller(在 Android 中安装数字签名,被调用)
|-- Contacts(拨号(调用)、联系人、通话记录)
|-- DeskClock(桌面时钟)
||-- Email(电子邮件)
||-- Gallery(相册,和 Camera 类似,多了列表)
||-- Gallery3D(3D 相册)
||-- GlobalSearch(为 Google 搜索服务,提供底层应用)
||-- GoogleSearch(Google 搜索)
||-- HTMLViewer(浏览器附属界面,被浏览器应用调用,同时提供存储记录功能)
||-- IM(即时通讯,为手机提供信号发送、接收、通信的服务)
||-- Launcher(登录启动项,显示图片框架等图形界面)
||-- Launcher2(登录启动项,负责应用的调用)
||-- Mms(彩信业务)
||-- Music(音乐播放器)
||-- PackageInstaller(安装、卸载程序的响应)
||-- Phone(电话拨号程序)
|-- Provision(预设应用的状态,使能应用)
||-- Settings(开机设定,包括电量、蓝牙、设备信息、界面、wifi 等)
||-- SoundRecorder(录音机,可计算存储所需空间和时间)
||-- Stk(接收和发送短信)
||-- Sync(空)
||-- Updater(空)
```

```
| '-- VoiceDialer(语音识别通话)
|-- inputmethods(输入法)
| |-- LatinIME(拉丁文输入法)
| |-- OpenWnn(OpenWnn 输入法)
| '-- PinyinIME(拼音输入法)
|-- providers(提供器,提供应用程序、界面所需的数据)
| |-- ApplicationsProvider(应用程序提供器,提供应用程序启动项、更新等)
| |-- CalendarProvider(日历提供器)
|-- ContactsProvider(联系人提供器)
| |-- DownloadProvider(下载管理提供器)
| |-- DrmProvider(创建和更新数据库时调用)
| |-- GoogleContactsProvider(联系人提供器的子类,用以同步联系人)
| |-- GoogleSubscribedFeedsProvider(设置信息提供器)
| |-- ImProvider(空)
| |-- ManagementProvider(空)
| |-- MediaProvider(媒体提供器,提供存储数据)
| |-- TelephonyProvider(彩信提供器)
| |-- UserDictionaryProvider(用户字典提供器,提供用户常用字字典)
| '-- WebSearchProvider(空)
|-- services(服务项目)
| |-- EasService(空)
| '-- LockAndWipe(空)
'-- wallpapers(墙纸)
|-- Basic(基本墙纸,系统内置墙纸)
|-- LivePicker(选择动态壁纸)
|-- MagicSmoke(壁纸特殊效果)
'-- MusicVisualization(音乐可视化,图形随音乐而变化)
prebuilt 目录(x86 和 arm 架构下预编译的一些资源)
|-- android-arm(arm-android 相关)
| |-- gdbserver(gdb 调试器)
| '-- kernel(模拟的 arm 内核)
|-- android-x86(x86-android 相关)
| '-- kernel(空)
|-- common(通用编译好的代码,是 Java 的)
|-- darwin-x86(drawin x86 平台)
| '-- toolchain(工具链)
| |-- arm-eabi-4.2.1
| |-- arm-eabi-4.3.1
| '-- arm-eabi-4.4.0
```

```
|-- darwin-x86_64(drawin x86 64bit 平台)
|-- linux-x86(Linux x86 平台)
| '-- toolchain(工具链,我们主要用这个)
| |-- arm-eabi-4.2.1
| |-- arm-eabi-4.3.1
| |-- arm-eabi-4.4.0
| '-- i686-unknown-linux-gnu-4.2.1(x86 版编译器)
|-- linux-x86_64(Linux x86 64bit 平台)
|-- windows(Windows 平台)
'-- windows-x86_64(64bit Windows 平台)
SDK
sdk 及模拟器
system 目录(底层文件系统库、应用及组件——C 语言)
|-- Bluetooth(蓝牙相关)
|-- core(系统核心工具盒接口)
| |-- adb(adb 调试工具)
| |-- cpio(cpio 工具,创建 img)
| |-- debuggerd(调试工具)
| |-- fastboot(快速启动相关)
| |-- include(系统接口头文件)
| |-- init(init 程序源代码)
| |-- libacc(轻量级 C 编译器)
| |-- libctest(libc 测试相关)
| |-- libcutils(libc 工具)
| |-- liblog(log 库)
| |-- libmincrypt(加密库)
| |-- libnetutils(网络工具库)
| |-- libpixelflinger(图形处理库)
| |-- libsysutils(系统工具库)
| |-- libzipfile(zip 库)
| |-- logcat(查看 log 工具)
| |-- logwrapper(log 封装工具)
| |-- mkbootimg(制作启动 boot.img 的工具盒脚本)
| |-- netcfg(网络配置 netcfg 源码)
| |-- nexus(google 最新手机的代码)
| |-- rootdir(rootfs,包含一些 etc 下的脚本和配置)
| |-- sh(shell 代码)
```

```
||-- toolbox(toolbox,类似 busybox 的工具集)
|'-- vold(SD 卡管理器)
|-- extras(额外工具)
||-- latencytop(a tool for software developers,identifying system latency happen)
||-- libpagemap(pagemap 库)
||-- librank(Java Library Ranking System 库)
||-- procmem(pagemap 相关)
||-- procrank(Java Library Ranking System 相关)
||-- showmap(showmap 工具)
||-- showslab(showslab 工具)
||-- sound(声音相关)
||-- su(su 命令源码)
||-- tests(一些测试工具)
|'-- timeinfo(时区相关)
'-- wlan(无线相关)
'-- ti(ti 网卡相关工具及库)
vendor 目录(厂家定制内容)
|-- aosp(android open source project)
|'-- products(一些板级规则)
|-- htc(HTC 公司)
||-- common-open(通用部分)
|'-- akmd(解压 img 用的工具)
||-- dream-open(G1 开放部分)
||-- prebuilt-open(预编译开放部分)
|'-- sapphire-open(sapphire 这款型号开放内容)
|-- pv-open(空)
|-- qcom(空)
'-- sample(google 提供的样例)
|-- apps(应用)
||-- client(用户)
|'-- upgrade(升级)
|-- frameworks(框架)
|'-- PlatformLibrary(平台库)
|-- products(产品)
|-- sdk_addon(sdk 添加部分)
'-- skins(皮肤)
'-- WVGAMedDpi(WVGA 适用的图片)
```

4.3 Android 源码简要分析

4.3.1 Android 必需的工具

1. Eclipse 平台

Eclipse 是一个运行插件的平台。用户应该安装 Eclipse Classic 的最新版本(本教程使用 V3.3.1),并按照安装 Android SDK 中的说明安装 Android Developer Tools(Eclipse 插件)。

2. 源代码

(1) AndroidManifest.xml(片段):这个文件是 Android 的应用程序部署描述符。

(2) IntentReceiver 类:演示 IntentReceiver 的实现,这个类处理 AndroidManifest.xml 文件中 IntentFilter 标记所公布的 intent。

(3) SaySomething.java:实现一个 Android 活动,这是本教程的示例应用程序的主要入口点。

(4) Main.xml:这个文件包含 Android 活动所用的视觉元素或资源。

(5) R.java:这个文件是由 Android Developer Tools 自动生成的,它把视觉资源"连接"到 Java 源代码。

(6) AndroidManifest.xml(完整):这是完整的 AndroidManifest.xml 文件,包含每个重要元素的描述。

(7) MobileServiceCallContacts.java:这个文件包含的代码显示联系人并对用户输入做出反应,执行 Google Maps 地址查找。

3. Android 术语

在 Eclipse 环境中开发 Android 应用程序需要了解 Eclipse 环境和 Android 平台的知识,了解以下术语会有助于用 Eclipse 插件开发 Android 应用程序。

1) Android

这是 Open Handset Alliance 的主打产品。它是一种针对移动设备的开放源码操作环境。

2) 模拟器

模拟另一个系统的软件工具。它常常是在 PC(IBM、Mac、Linux)上运行的一个环境模拟另一个环境,例如移动计算设备。

3) Linux

一种开放源码的操作系统内核,许多计算平台都使用这种操作系统,包括服务器、桌面计算机、网络设备和移动计算设备。Android 在 Linux 内核上运行。

4) Dalvik Virtual Machine

Dalvik VM 是 Android 产品组合中的一种操作环境,它在运行时解释应用程序代码。Dalvik VM 与 JVM 相似,但两者不兼容。

5) Java 编程环境

工具集,包括编译器、资源编译器、调试器、模拟器以及用来运行应用程序的 JVM。

6) intent

intent 是一种构造,应用程序可以通过它发出请求,就像是发出求助信号。intent 可能

像下面这样：

(1) Wanted：An application to help me look up a contact。

(2) Wanted：An application to help me display this image。

(3) Wanted：An application to perform this geographic-based search。

应用程序可以按照相似或互补的方式进行注册，表明它们有能力或有兴趣执行各种请求或 intent。例如：

(1) Available：Application ready and willing to present contact records in clear, concise manner。

(2) Available：Application ready and willing to perform a geographic search。

以上这些是 IntentFilter 的示例。

7) IntentFilter

应用程序通过一个称为 IntentFilter 的构造声明它们能够执行某些类型的操作。IntentFilter 可以在运行时进行注册，也可以在 AndroidManifest.xml 文件中设置。下面的清单 1 所示代码片段取自一个对 SMS(文本)消息进行响应的 Android 应用程序。

清单 1：对 SMS 进行响应的 Android 应用程序。

```
<receiver class = ".MySMSMailBox" >
          <intent - filter>
            <action
android:value = "android.provider.Telephony.SMS_RECEIVED" />
            </intent - filter>
      </receiver>
```

4.3.2 Android 应用程序概述

本节主要介绍 Android 应用程序的 4 种主要类型：活动、服务、接收器和 ContentProvider。同时还要介绍显示用户界面(UI)元素的视图。

1. 活动

活动(Activity)是最常用的 Android 应用程序形式。活动在一个称为视图的类的帮助下为应用程序提供 UI。视图类实现各种 UI 元素，例如文本框、标签、按钮和计算平台上常见的其他 UI 元素。一个应用程序可以包含一个或多个活动，这些活动通常与应用程序中的屏幕形成一对一的关系。

应用程序通过调用 startActivity()或 startSubActivity()方法从一个活动转移到另一个活动。如果应用程序只需切换到新的活动就应该使用前一个方法；如果需要异步地调用/响应模式就使用后一个方法。在这两种情况下都需要通过方法的参数传递一个 intent，由操作系统负责决定哪个活动最适合满足指定的 intent。

2. 服务和接收器

与其他多任务计算环境一样，在后台运行着一些应用程序，它们执行各种任务。Android 把这种应用程序称为服务(Service)。服务是没有 UI 的 Android 应用程序。

接收器(Receiver)是一个应用程序组件，它接收请求并处理 intent。与服务一样，接收器在一般情况下也没有 UI 元素。接收器通常在 AndroidManifest.xml 文件中注册。下面

的清单 2 所示代码是接收器代码的示例(注意,接收器的类属性是负责实现这个接收器的 Java 类)。

清单 2:接收器代码。

```
package com.msi.samplereceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentReceiver;
public class myreceiver extends IntentReceiver
{
  public void onReceiveIntent(Context arg0, Intent arg1)
  {
    // do something when this method is invoked.
  }
}
```

3. 用 ContentProvider 进行数据管理

ContentProvider 是 Android 的数据存储抽象机制。例如移动设备上常见的一种数据如地址簿或联系人数据库,地址簿包含所有联系人及其电话号码,用户在使用手机时可能需要使用这些数据。ContentProvider 对数据存储的访问方法进行抽象,它在许多方面起到数据库服务器的作用。对数据存储中数据的读写操作应该通过适当的 ContentProvider 传递,而不是直接访问文件或数据库。可能还有 ContentProvider 的"客户机"和"实现"。

4. 视图

Android 活动通过视图(View)显示 UI 元素。视图采用以下布局设计之一。

1) LinearVertical

后续的每个元素都排在前一个元素下面,形成一个单一列。

2) LinearHorizontal

后续的每个元素都排在前一个元素右边,形成一个单一行。

3) Relative

后续的每个元素相对于前一个元素有一定的偏移量。

4) Table

与 HTML 表相似的一系列行和列,每个单元格可以包含一个视图元素。

选择一种布局(或布局的组合)之后,就可以用各个视图显示 UI。

视图元素由大家熟悉的 UI 元素组成,包括:

(1) Button。

(2) ImageButton。

(3) EditText。

(4) TextView(与标签相似)。

(5) CheckBox。

(6) Radio Button。

(7) Gallery 和 ImageSwitcher(用来显示多个图像)。

(8) List。

(9) Grid。

(10) DatePicker。

(11) TimePicker。

(12) Spinner(与组合框相似)。

(13) AutoComplete(具有文本自动补全特性的 EditText)。

视图是在一个 XML 文件中定义的。清单 3 所示的代码给出一个简单的 LinearVertical 布局示例。

清单 3: 简单的 LinearVertical 布局。

```
<?xml version = "1.0" encoding = "utf - 8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "fill_parent"
    android:layout_height = "fill_parent"
    >
<TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "Activity 1!"
    />
<TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "Activity 1, second text view!"
    />
<Button
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "Switch To Activity 2"
  id = "@ + id/switchto2"
  />
</LinearLayout>
```

4.3.3 构建 SaySomething Android 应用程序

本节使用 Android Developer Tools 创建一个名为 SaySomething 的基本应用程序,创建之后再对其进行调试和运行。

1. New Project 向导

先创建一个新项目,然后选择用来创建 Android 项目的向导。这个应用程序需要设置:

(1) Project name。

(2) Location。

(3) Package name。

(4) Activity name。可以认为这是应用程序的主表单或屏幕。

(5) Application name。

设置完成后的这个 Android 新项目如图 4-9 所示。



图 4-9　新的 Android 项目

单击 Finish 按钮，这会创建一个默认的应用程序，可以构建和运行它，也可以在 Package Explorer 中查看它的组件。

2. Package Explorer

Package Explorer（在 Eclipse 的 Java 透视图中）显示 Android 示例应用程序的所有组件，如图 4-10 所示。

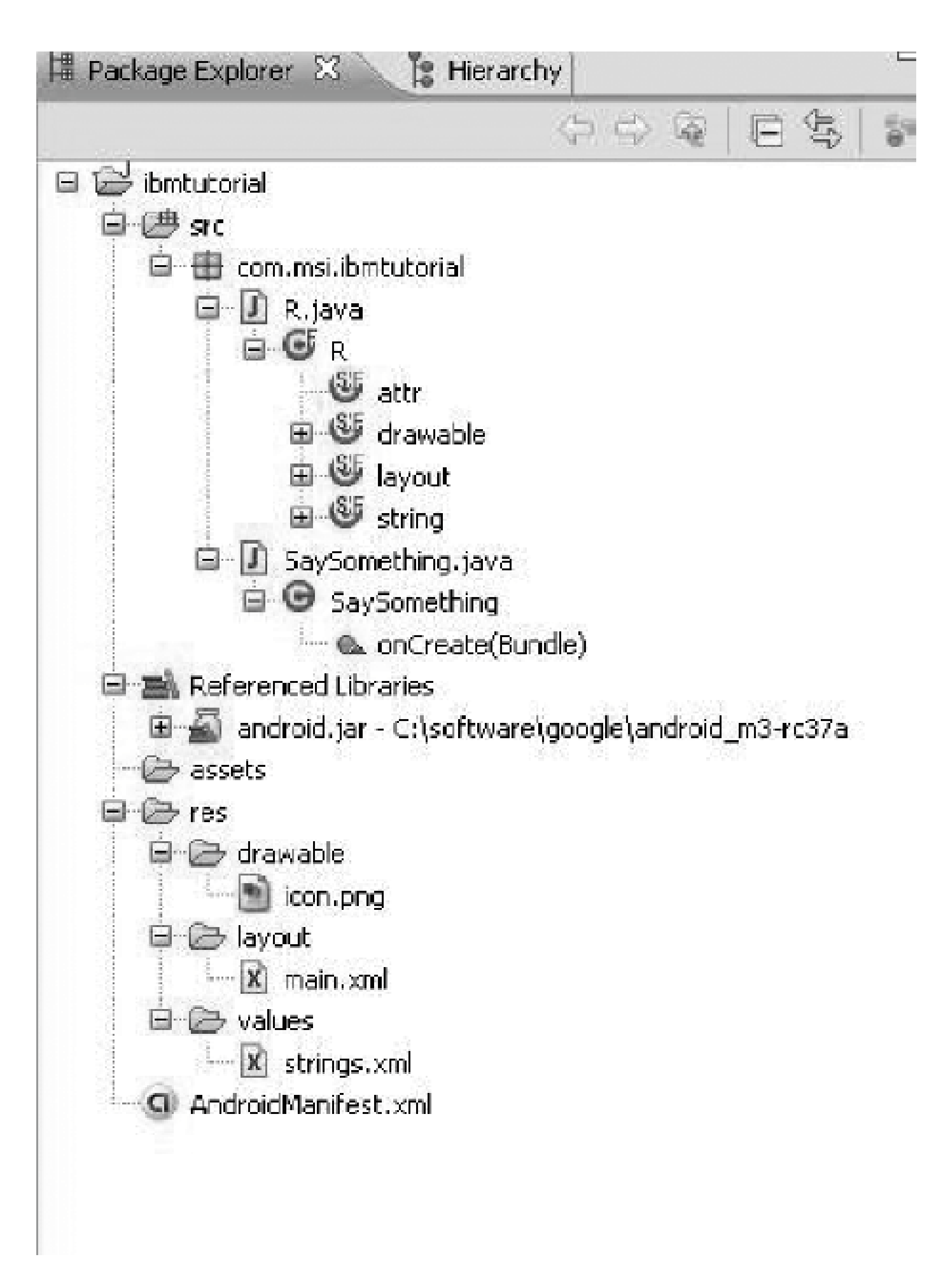


图 4-10　Package Explorer

在 Package Explorer 中(如图 4-10 所示)需要注意以下组件:

(1) src 目录。包含示例应用程序的包,即 com. msi. ibmtutorial。

(2) R. java 文件。Android Developer Tools 自动创建这个文件,它提供访问 Android 应用程序的各种资源所需的常量。

(3) SaySomething. java 文件。应用程序的主 Activity 类的实现。

(4) Referenced Libraries 目录。包含 android. jar 文件,这是 Android SDK 中的 Android 运行时类的 jar 文件。

(5) res 目录。包含应用程序的资源,例如图标、布局文件、字符串等。

(6) AndriodManifest. xml 文件。示例应用程序的部署描述符。

3. 源代码

1) 应用程序的主 Activity

这个示例应用程序由一个 Activity 组成,即 SaySomething。正如前面提到的,SaySomething 类是在 SaySomething. java 文件中实现的,程序代码如清单 4 所示。

清单 4:SaySomething. java。

```
package com.msi.ibmtutorial;
import android.app.Activity;
import android.os.Bundle;
public class SaySomething extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle)
    {
        super.onCreate(icicle);
        setContentView(R.layout.main);
    }
}
```

在这段源代码片段中要注意几点:

(1) SaySomething 是一个普通的 Java 类,包含包和导入语句。

(2) SaySomething 扩展 android. app 包中的 Android 基类 Activity。

(3) onCreate()方法是这个活动的入口点,它接受一个 Bundle 类型的参数。Bundle 类本质上是 map 或 hashmap 的包装器,在这个参数中传递构造活动所需的元素。

(4) setContentView()方法负责用 R. layout. main 参数创建主 UI。R. layout. main 是应用程序资源中主布局的标识符。

2) 应用程序的资源

正如前面提到的,Android 中的资源放在项目的 res 子目录中。资源分为三类(如图 4-9 所示):

(1) drawable。这个目录包含图形文件,例如图标(icon. png)和位图。

(2) layout。这个目录包含表示应用程序布局和视图的 XML 文件,例如 main. xml。

(3) values。这个目录包含 strings. xml 文件,这是为应用程序实现字符串本地化的主

要方法。

接下来介绍 main. xml 文件，了解示例应用程序的 UI 资源。

3）main. xml 文件

这个示例应用程序包含一个 Activity 和一个 View。应用程序包含一个名为 main. xml 的文件，它代表 Activity 的主 UI 的视觉方面。注意在 main. xml 中没有指定在哪里使用这个布局，这意味着如果需要的话可以在多个 Activity 中使用它。清单 5 所示程序代码给出布局文件的内容。

清单 5：布局文件。

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "fill_parent"
    android:layout_height = "fill_parent"
    >
< TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "Hello World, SaySomething"
    />
</LinearLayout >
```

这是最简单的布局，其中只有一个垂直的线性布局，这意味着所有元素排成一列；这里有一个 TextView（代表不可编辑的静态文本）元素，它与其他开发环境中的标签类似；每个 View 元素（例如这个示例中的 LinearLayout 和 TextView）都有属于 Android 名称空间的属性，一些属性是所有 View 元素都有的，例如 android：layout_width 和 android：layout_height，这些属性可以采用的值是 fill_parent，使 View 元素占满可用的空间，也可以认为这就是拉伸；wrap_content 这个值让 Android 把元素一个接一个地排列，不进行拉伸。

Android 在应用程序构建期间对所有资源进行编译，编译过程的输出之一是 R. java 文件，这个文件向应用程序的其余部分提供资源。下面讨论 R. java 文件。

4）R. java 文件

R. java 文件是在 Android 构建应用程序时自动创建的，所以不要手工修改它，因为所有修改都会丢失。清单 6 所示程序代码给出了这个示例应用程序的 R. java 文件。

清单 6：R. java 文件。

```
/* AUTO - GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */
package com.msi.ibmtutorial;
public final class R {
    public static final class attr {
```

```
        }
        public static final class drawable {
            public static final int icon = 0x7f020000;
        }
        public static final class layout {
            public static final int main = 0x7f030000;
        }
        public static final class string {
            public static final int app_name = 0x7f040000;
        }
    }
```

R 类包含一些匿名子类，每个子类包含前面描述的各种资源的标识符，注意这些类都是静态的；R. layout. main 这个标识符代表由 main. xml 定义的布局，在活动的 onCreate 方法中使用过这个值：

```
setContentView(R.layout.main);
```

这就是在运行时把特定的活动（在这个示例中是 SayAnything）和特定的布局（主布局）联系在一起的方法。

4. 构建应用程序

1）Problems 面板

在默认情况下，每次保存文件时都将对它们进行编译，如图 4-11 所示。

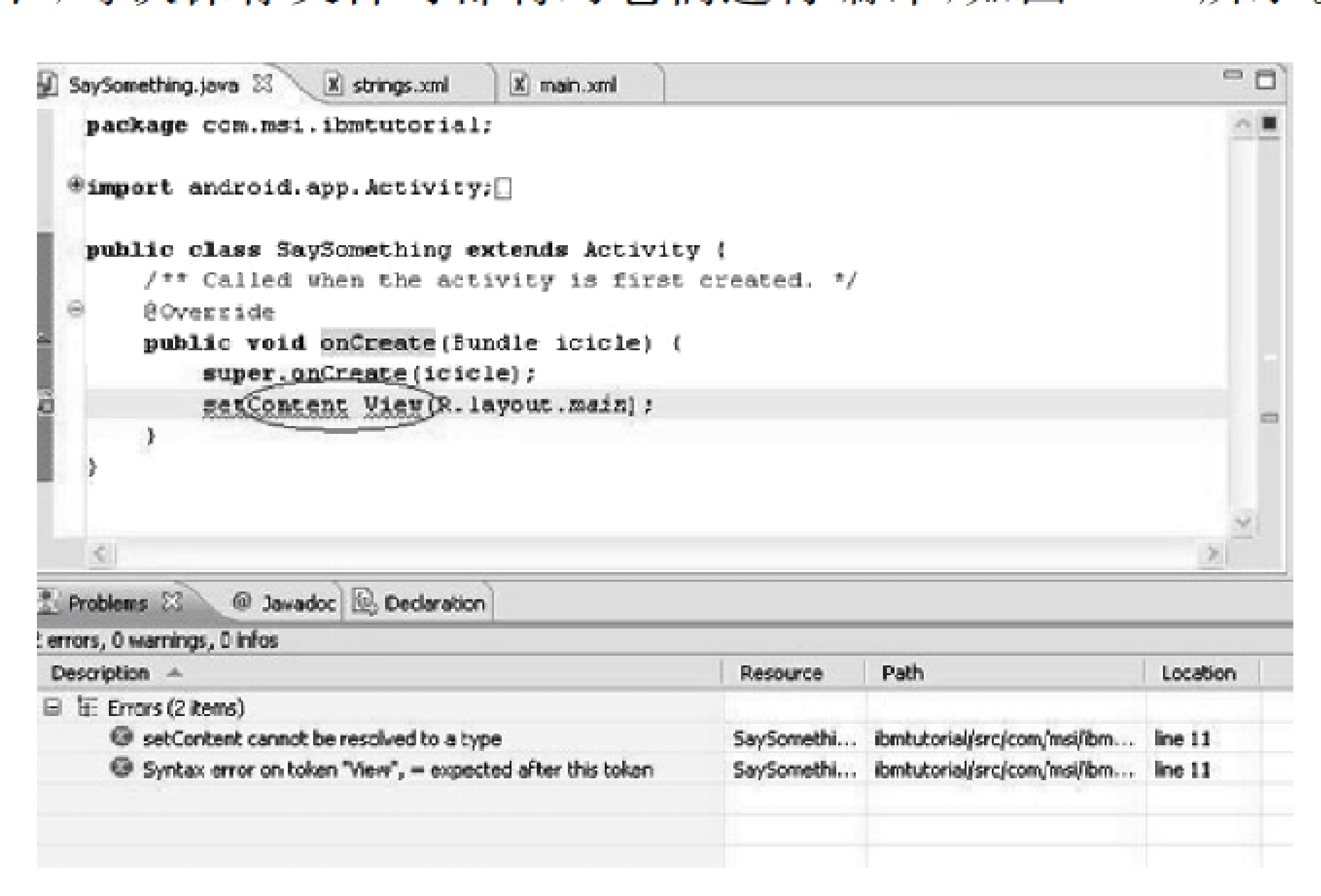


图 4-11　Problems 面板

这里在源代码中引入了一个错误，即在 setContent 和 View 之间加了一个空格，在保存这个文件时，它被编译并在屏幕底部的 Problems 面板中显示错误。在源代码中纠正这个错误之后，应用程序就能够成功编译，并从问题列表中删除错误。

2）AndroidManifest. xml 文件

AndroidManifest. xml 文件是 Android 应用程序的部署描述符。这个文件列出应用程序中包含的所有活动、服务、内容提供器和接收器，以及应用程序支持的 IntentFilter。下面的清单 7 所示程序代码是这个示例应用程序的完整的 AndroidManifest. xml 文件。

清单 7：AndroidManifest. xml 文件。

```
<?xml version = "1.0" encoding = "utf - 8"?>
<manifest xmlns:android = "http://schemas.android.com/apk/res/android"
    package = "com.msi.ibmtutorial">
    <application android:icon = "@drawable/icon">
        <activity class = ".SaySomething" android:label = "@string/app_name">
            <intent - filter>
                <action android:value = "android.intent.action.MAIN" />
                <category android:value = "android.intent.category.LAUNCHER" />
            </intent - filter>
        </activity>
    </application>
</manifest>
```

请注意以下情况：

(1) 这里指定了源文件中的包名，采用了与 Java 源文件和导入语句相似的模式。<manifest>标签的实际作用是导入这个包中的类。在这个文件中，所有非完全限定的类都属于 package 属性指定的包。

(2) <application>标签的一个属性引用了应用程序的一个资源。请注意 drawable 标识符前面的@符号的意思是，在应用程序资源的 drawable 目录中寻找名为 icon 的资源。

(3) <activity>标签包含以下属性和值：

① class 属性表示实现这个 Activity 的 Java 类。

② android:label 是应用程序的名称，它来自一个字符串资源。string. xml 文件包含应用程序的本地化字符串。

③ <intent-filter>表示应用程序中可用的 IntentFilter。这是 Android 应用程序中最常见的 IntentFilter。这个过滤器的实际意思是，它实现主操作(也就是入口点)，而且它位于 OS 的启动器中。这意味着可以在 Android 设备上像启动其他应用程序一样从应用程序主列表中启动它。

5. 启动应用程序

应用程序已经成功地编译了，现在运行这个示例应用程序，即在 Eclipse 中的 Android Emulator 上启动该应用程序。

(1) 在 Eclipse 中执行 Run→Open Run Dialog 菜单命令或单击工具栏上的快捷按钮，打开 Run 对话框。选择 Android Application 选项并单击 New configuration 的图标，进而创建启动配置，如图 4-12 所示。

(2) 选择 Project 为 ibmtutorial，在 Activity 下拉列表中选择启动 com. msi. ibmtutorial. SaySomething，单击 Apply 按钮。

(3) 选择 Run 对话框中的 Emulator 选项卡，根据需要指定 Emulator 设置，如图 4-13 所示。

这里有几种可供选择的屏幕大小和方向，还有网络选择。如果运行应用程序的移动设备的 Internet 连接速度不同，则网络选择就很重要了，在构造应用程序原型时，选择 Full(完整网络速度)而且没有延迟。开发了主要功能之后，最好在比较真实的网络环境中进行测试，看看应用程序的响应速度如何。

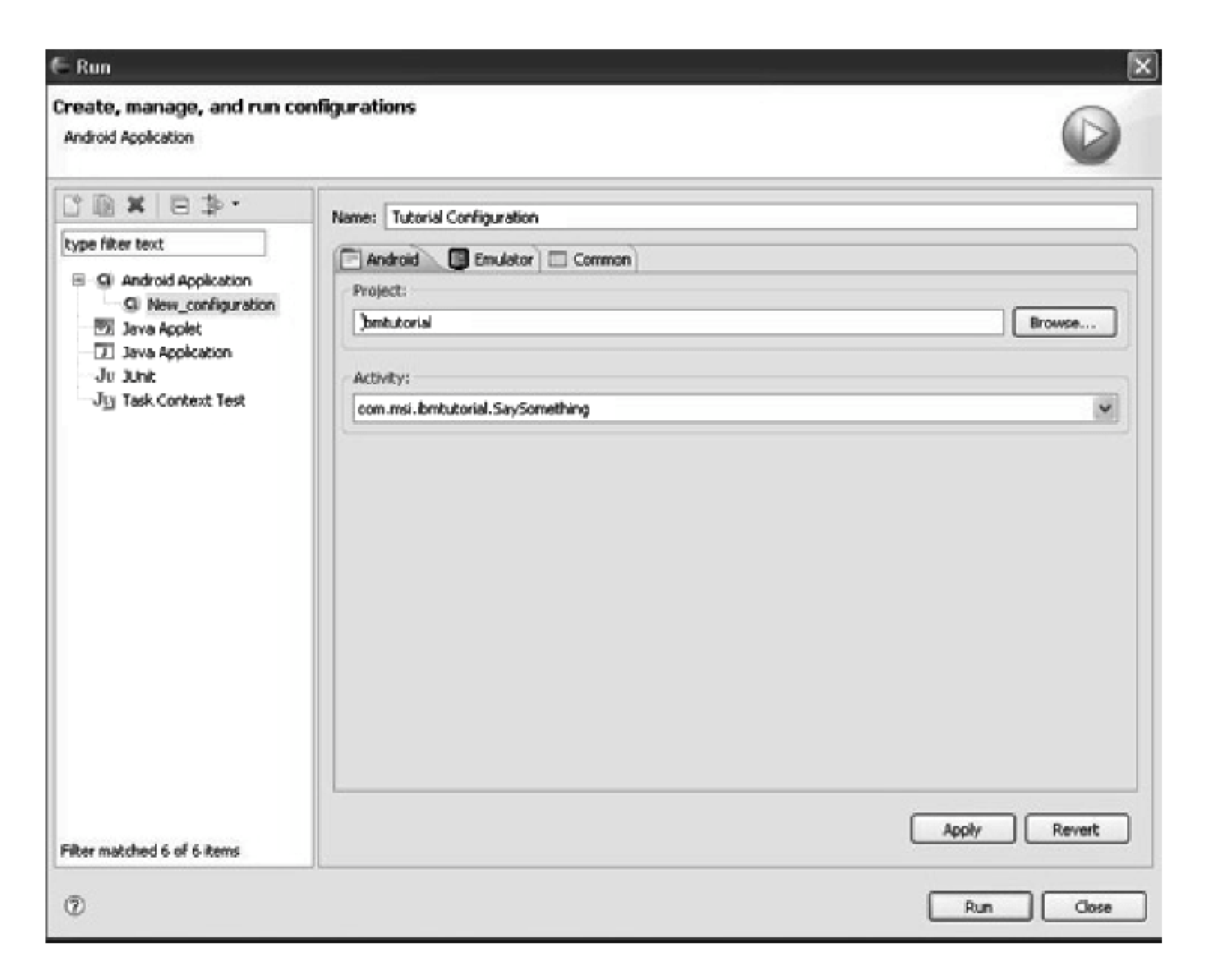


图 4-12　Run 对话框

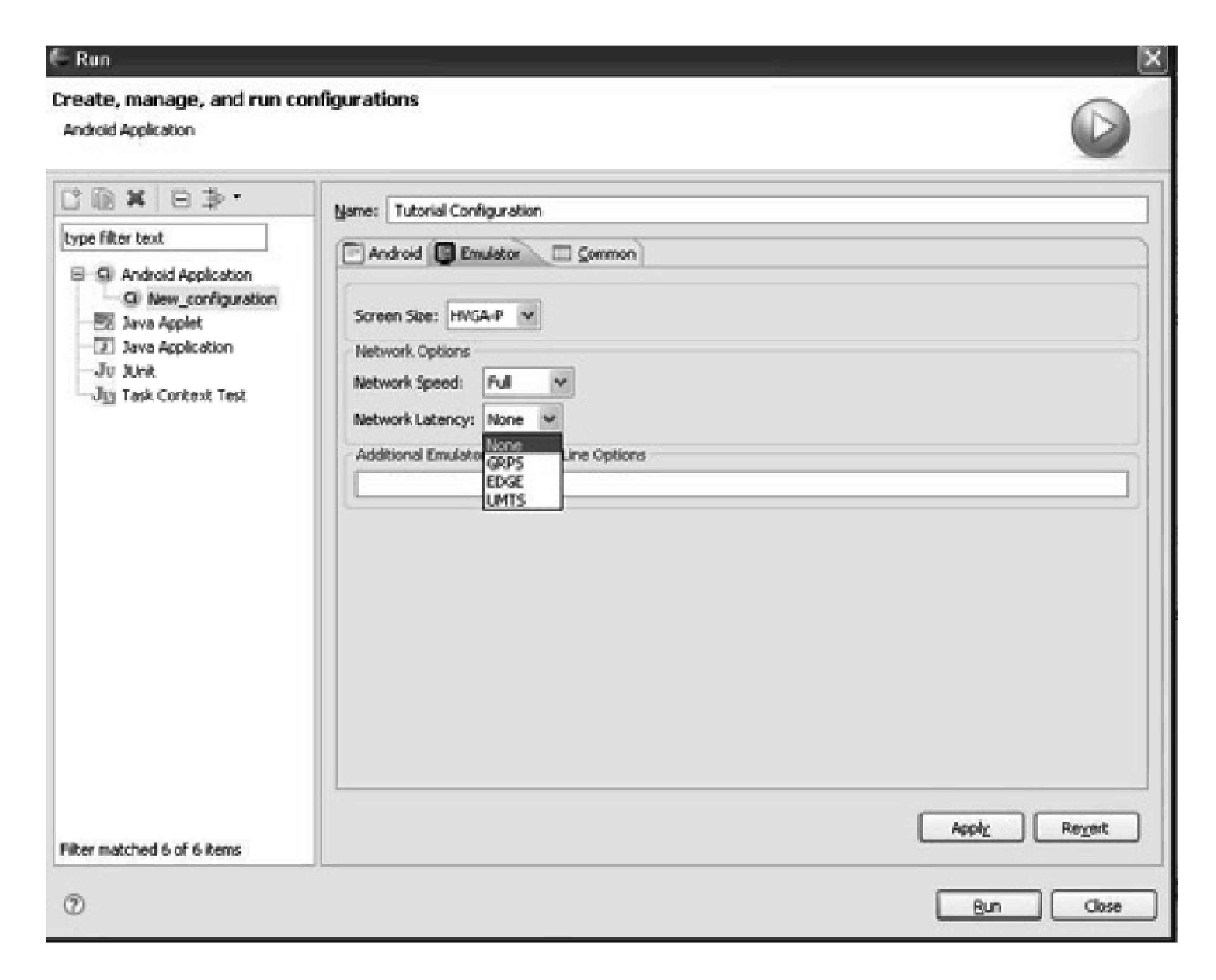


图 4-13　Run 对话框中的 Emulator 选项卡

(4) 单击图 4-13 中的 Apply 按钮，单击 Run 按钮运行示例应用程序。运行结果如图 4-14 所示。

应用程序已经在 Emulator 上运行了，现在看看幕后发生的情况。DDMS(Dalvik Debug Monitor Service)将会帮助检查应用程序的运行情况。



图 4-14　运行 SaySomething 示例应用程序

6. 调试应用程序

要想检查正在运行的应用程序中发生了什么情况，就需要查看正在运行的 Dalvik VM。在 Eclipse 中执行 Window→Open Perspective→Other 菜单命令，在出现的对话框中选择 DDMS 选项。这会在 Eclipse 中打开一个新的透视图，其中有许多有趣的窗口。下面简要介绍一下 DDMS 透视图中提供的资源：

（1）LogCat 是一个日志文件，它记录 Dalvik VM 中发生的 Activity。应用程序可以通过命令“Log. i(tag, message);”在这个日志文件中添加自己的日志项，其中的 tag 和 message 都是 Java 字符串。Log 类属于 android. util. Log 包。图 4-15 所示为 LogCat。

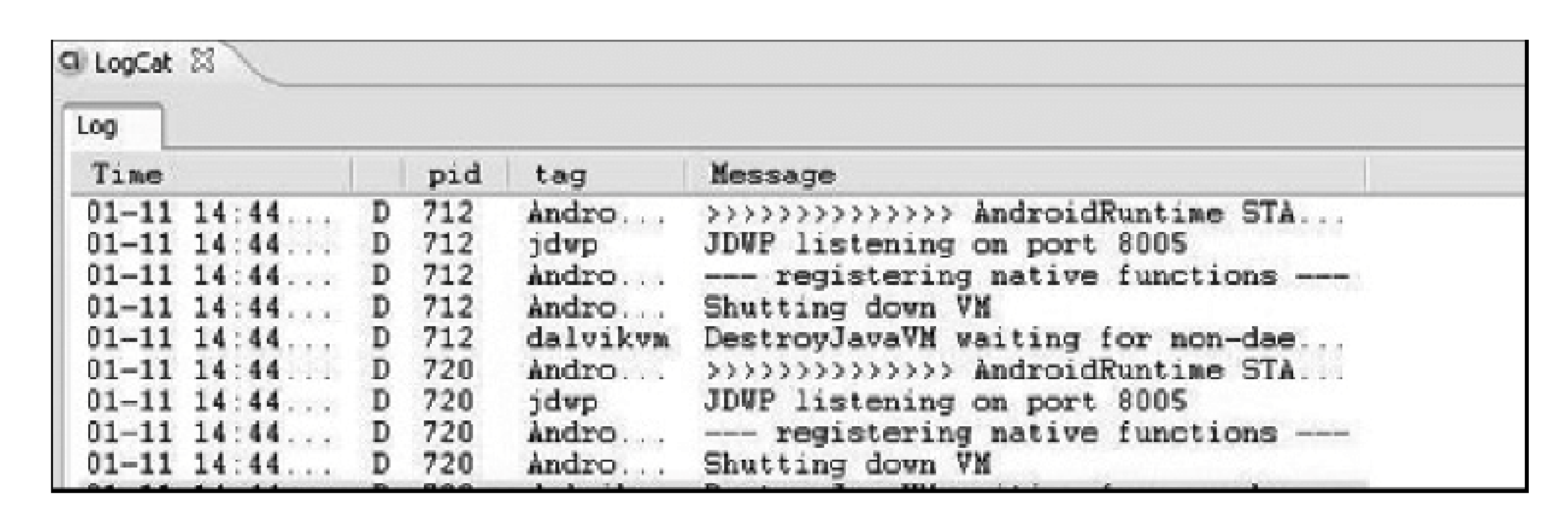


图 4-15　LogCat

（2）DDMS 中另一个方便的工具是文件管理器，可以用它访问 Emulator 的文件系统。在模拟器上部署本教程示例应用程序的位置如图 4-16 所示。

用户应用程序部署在/data/app 目录中，而 Android 内置的应用程序部署在/system/app 目录中。

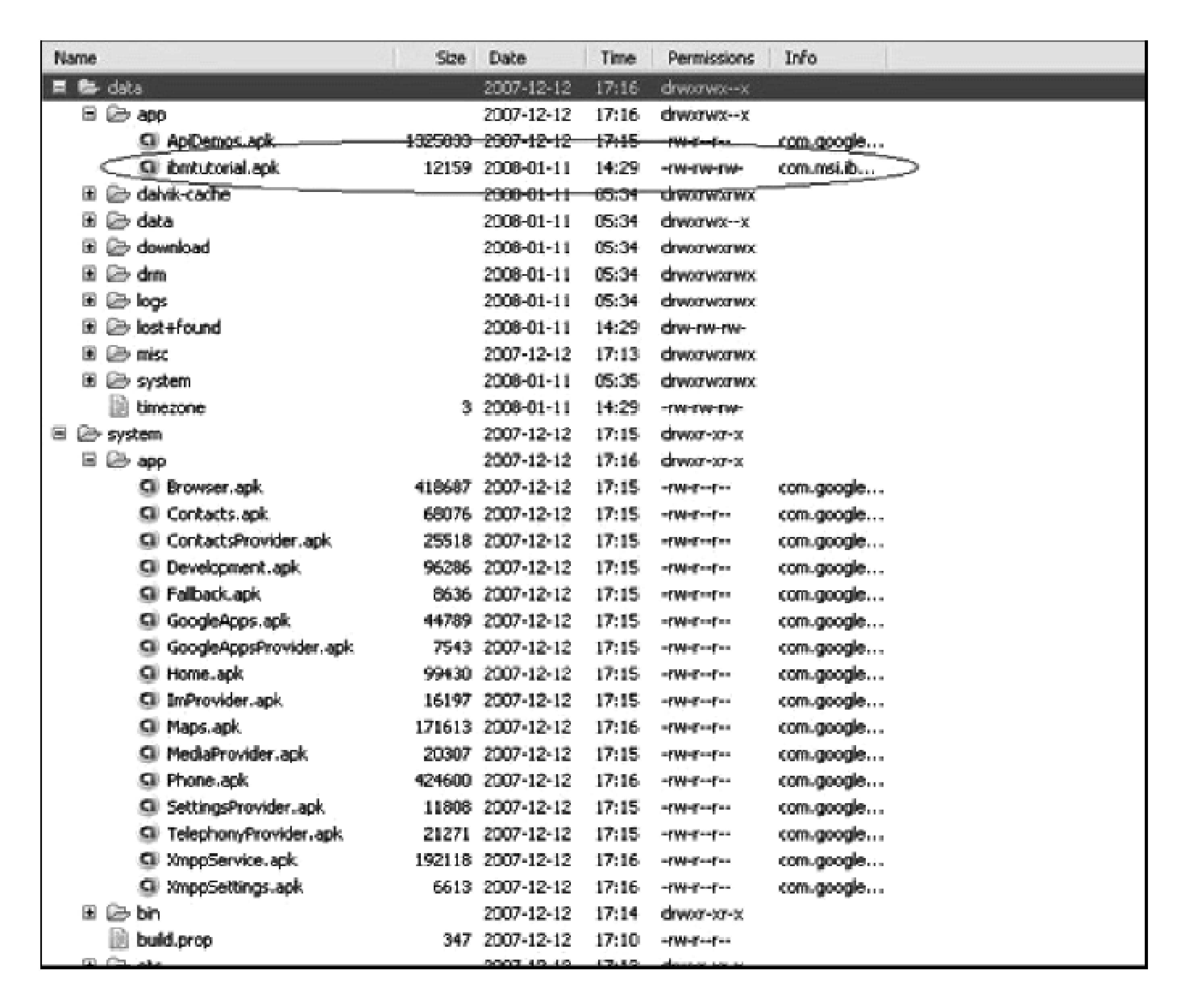


图 4-16　在 Emulator 上部署的示例应用程序

(3) 在 DDMS 中还可以查看正在运行的进程，如图 4-17 所示。

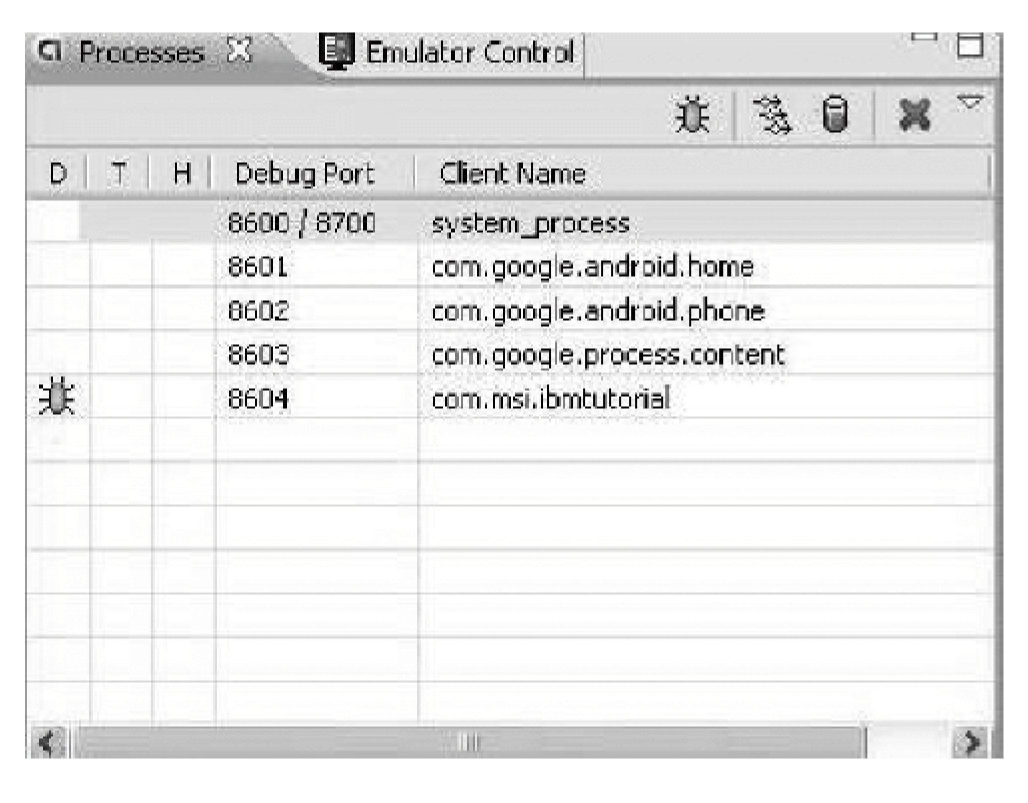


图 4-17　正在运行的进程列表

4.3.4　创建内容提供器和 Google Maps 应用程序

上一节已经创建了一个完整的应用程序示例，现在简要讨论一下更加复杂的应用程序。内容提供器和 Google Maps 应用程序适用于提供上门服务的专业人员，例如设备维修技术人员必须找到去客户地址的路线。这个应用程序使用 Android 内置的联系人数据库作为记录存储库。下面讲解如何访问内容提供器中的数据，还将看到 intent 的效果。这里将用联系人数据库中的地址数据执行 Google Maps 搜索。

(1) 为了让这个应用程序在 Android Emulator 上正确运行，必须记录一些联系人，而且必须填写家庭地址字段。图 4-18 所示为 Emulator 中与联系人应用程序相关的条目。

(2) 下面的清单 8 所示的程序代码是这个应用程序的一个代码片段。注意，这个应用



图 4-18　Emulator 中与联系人应用程序相关的条目

程序的主 Activity 类扩展为 ListActivity，这是因为需要在列表中显示信息。

清单 8：第一个代码片段。

```
public class MobileServiceCallContacts extends ListActivity
{
     final String tag = "MSCC";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle)
    {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        // Get a cursor with all people
        Cursor c = getContentResolver().query(People.CONTENT_URI, null, null,null, null);
        startManagingCursor(c);

        ListAdapter adapter = new SimpleCursorAdapter(this,android.R.
        layout.simple_list_item_1,c,new String[] {People.NAME} ,new int[]
        {android.R.id.text1});
        setListAdapter(adapter);

    }
…
}
```

这里使用游标类查询联系人数据库，这个“结果集”游标通过 ListAdapter 类链接到 UI。图 4-19 所示为当设备上记录了联系人时应用程序的效果(这个屏幕上的记录没有排序)。



图 4-19　应用程序的运行效果

(3) 可以通过单击、按 Emulator 的中间按钮或按键盘上的 Enter 键选择一个联系人。选择联系人之后，程序必须查询所选联系人的地址，这要使用覆盖的 onListItemClick()方法。这个方法的实现有 4 个重要参数，其中最重要的是 dbidentifier。因为游标绑定到 UI，所以在调用这个方法时，它实际上会获得底层数据源的标识符。可以使用 dbidentifier 字段在联系人数据库中查询所需的信息，还可以使用它启动联系人应用程序，所用的 intent 见清单 9 所示程序代码中被注释掉的代码。

清单 9：覆盖的 onListItemClick()方法。

```
@Override
protected void onListItemClick(ListView list,View view,int position,long
                                                     dbidentifier)
{
     super.onListItemClick(list,view,position,dbidentifier);

     try
     {
          // this commented out code below will launch the Contacts application
     // and "view" the contact Intent myIntent = \
new Intent(android.content.
// Intent.VIEW_ACTION,new ContentURI("content://contacts/people/"
     // + dbidentifier)); startSubActivity(myIntent,position);


          // let's lookup specifics on this record
          ContentURI theContact = \
new ContentURI(android.provider.Contacts.ContactMethods.CONTENT_URI.toURI());
```

```
            // IMPORTANT
            // in order to use this sample application, you need to have at least
        // one Contact record on your Android emulator\
            // and be sure to have populated the 'Home Address field'
            //

            // this "where clause" is for HOME address and for the person record
        // selected in the GUI (id, dbidentifier)
            Cursor c = managedQuery(theContact,null," type = 1 and person = " +
                                    dbidentifier,null);

            if (!c.first())
            {
                showAlert("MSCC","No Contact Methods Available!","",true);
                return;
            }
            String address = c.getString(c.getColumnIndex("data"));

            address = address.replace("\n","");
            address = address.replace(",","");
            address = address.replace(" ","+");

            Intent geoIntent = new Intent("android.intent.action.VIEW",
                                 new ContentURI\
    ("geo:0,0?q=" + address));
            startActivity(geoIntent);
        }
        catch (Exception ee)
        {
            Log.i(tag,ee.getMessage());
        }
    }
```

(4) 找到地址之后，需要通过一些简单的字符串操作清理数据，准备查询 Google Maps。geoIntent 是一个用来执行地理搜索的新的 intent，它通过调用 Google Maps 提供默认的 Android Emulator 图像。

(5) 说明

上一节的应用程序的所有主要元素仍然适用于这个应用程序，它有一个从主应用程序屏幕启动的 Activity，当然还有 AndroidManifest.xml 文件（在 4.4 节中可以找到完整的源代码）；关于本节的应用程序还需要注意的是，AndroidManifest.xml 文件中包含一个条目，它使应用程序能够读取联系人数据库：

```
<uses-permission id="android.permission.READ_CONTACTS" />
```

如果不这样做，Linux 内核就会禁止应用程序访问联系人数据库。

4.4 Android 平台应用向 OMS 平台迁移

4.4.1 OMS 概述

OMS 全称是 Open Mobile System，即开放式手机操作系统。OMS 操作系统是中国移动主导的开放式操作系统，它基于 Google Android，是经过定制 Google Android 平台演变出的一种更适合我国用户的智能手机操作系统。OMS 基于 Linux 内核，采用 Android 源代码，在 Google 的 Android 系统基础上二次开发而来，通过大唐移动的 3G 芯片 TD-SCDMA 以 Modem AP 的方式桥接使 Android 平台兼容中国移动 TD 网络，在业务层和用户体验层与此前的 Google 手机完全不一样。OMS 是基于 Linux 2.6 之上运行的，提供核心系统服务：安全、内存管理、进程管理、网络组、驱动模型。Linux 内核部分相当于一个介于硬件层和系统中其他软件组之间的一个抽象层次。OMS 参考了苹果 iPhone 操作系统和 Windows Mobile 手机操作系统的优点，中国移动在 OMS 上嵌入所有自主开发或者协议方开发的应用软件，OMS 在终端手机上完整地深度定制了飞信、快讯、无线音乐随身听、139 邮箱、移动梦网、号薄管家、百宝箱等中国移动数据业务。

从 SDK 中的 OMS Emulator（如图 4-19 所示）可以看出，无论从外观、用户 UI、业务和一些网络服务，OMS 都做了全面的二次深度定制，对用户 UI 做了高度的二次开发；用户的交互体验质量得到了升级，更多的移动服务绑定到系统中，提供了全面而又快捷的通信服务；新增加的多媒体应用（网络电视、FM 调频收音、无线音乐等）会让用户对整个 OMS 有新的体验。OMS 从传统的手机中脱颖而出，将成为中国市场新 3G 的佼佼者。

4.4.2 OMS 特色

1. OMS UI 特色

从图 4-20 所示的 OMS Emulator 中可以看到，整体界面变换很大，给人一种耳目一新的感觉。下面来看看 OMS 在感官方面带来了哪些好的 UI 用户体验。

1）OMS 的启动界面

OMS 的启动界面如图 4-21 所示，它分为两个部分，第一部分（图 4-21 中所示的左半部分）中，启动时有“心机”字样和中国移动的标识，附带音乐铃声；第二部分（图 4-21 中所示的右半部分）出现 OPhone 标识，在之前最早出现的 OMS SDK 中，第二部分带有“O”字样的动画光芒效果。

2）OMS 桌面主题

OMS 提供了绚丽的主题。在主页中选择 Menu 菜单，在 Setting 中选中 Home Theme，可以看到有 3 种 Theme（主题），如图 4-22 所示。

（1）Classical 主题：是经典主题，图标呈卡片状，附带颜色，如图 4-22（a）所示。

（2）SummerCool 主题：是夏日清凉主题，图标呈金属黑半透明状，给人一种冷酷的感觉，如图 4-22（b）所示。

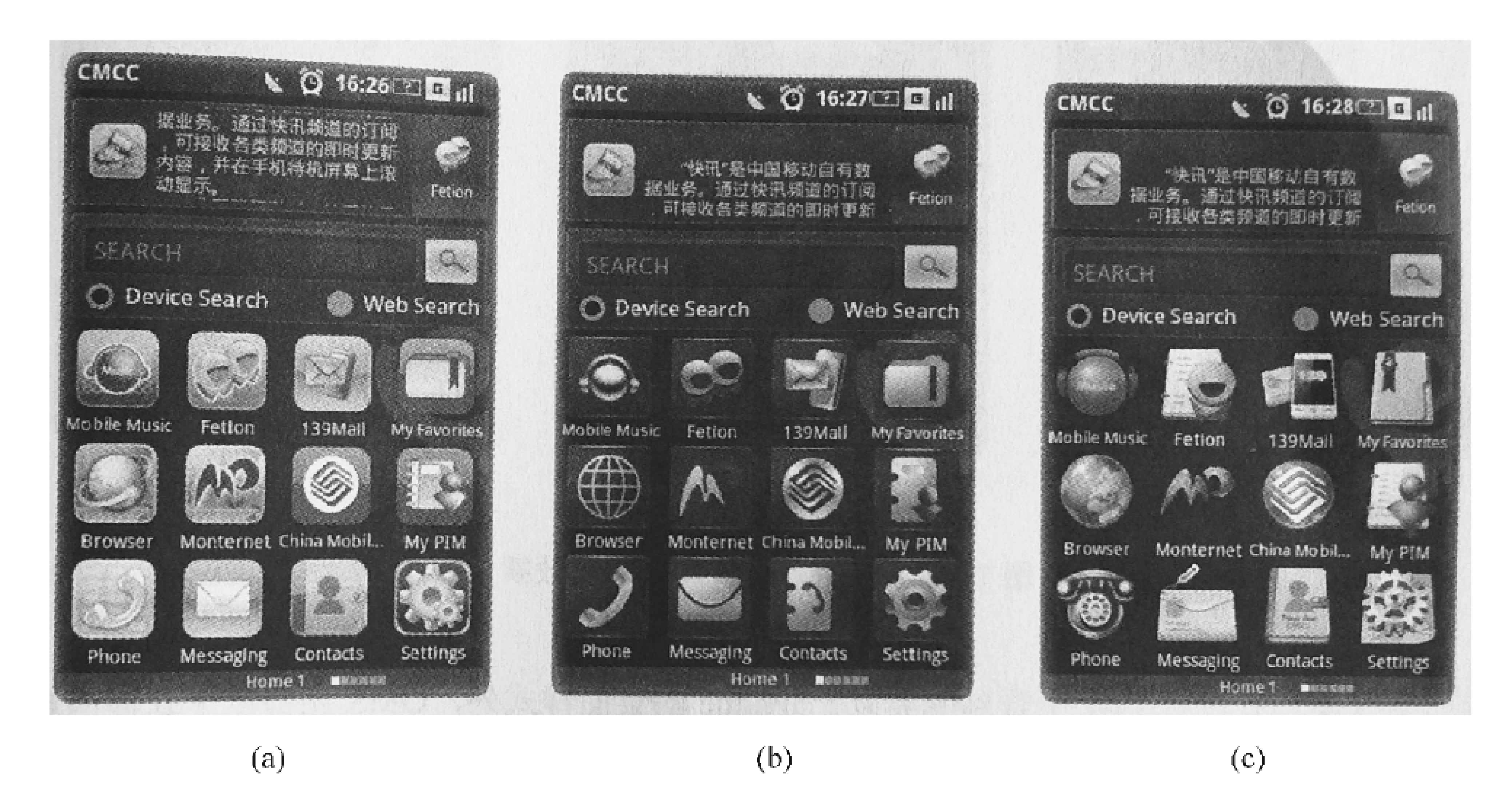
（3）3D 主题：图标呈立体状，有质感，给人一种身临其境的体验，如图 4-22（c）所示。



图 4-20　OMS Emulator



图 4-21　OMS 启动界面



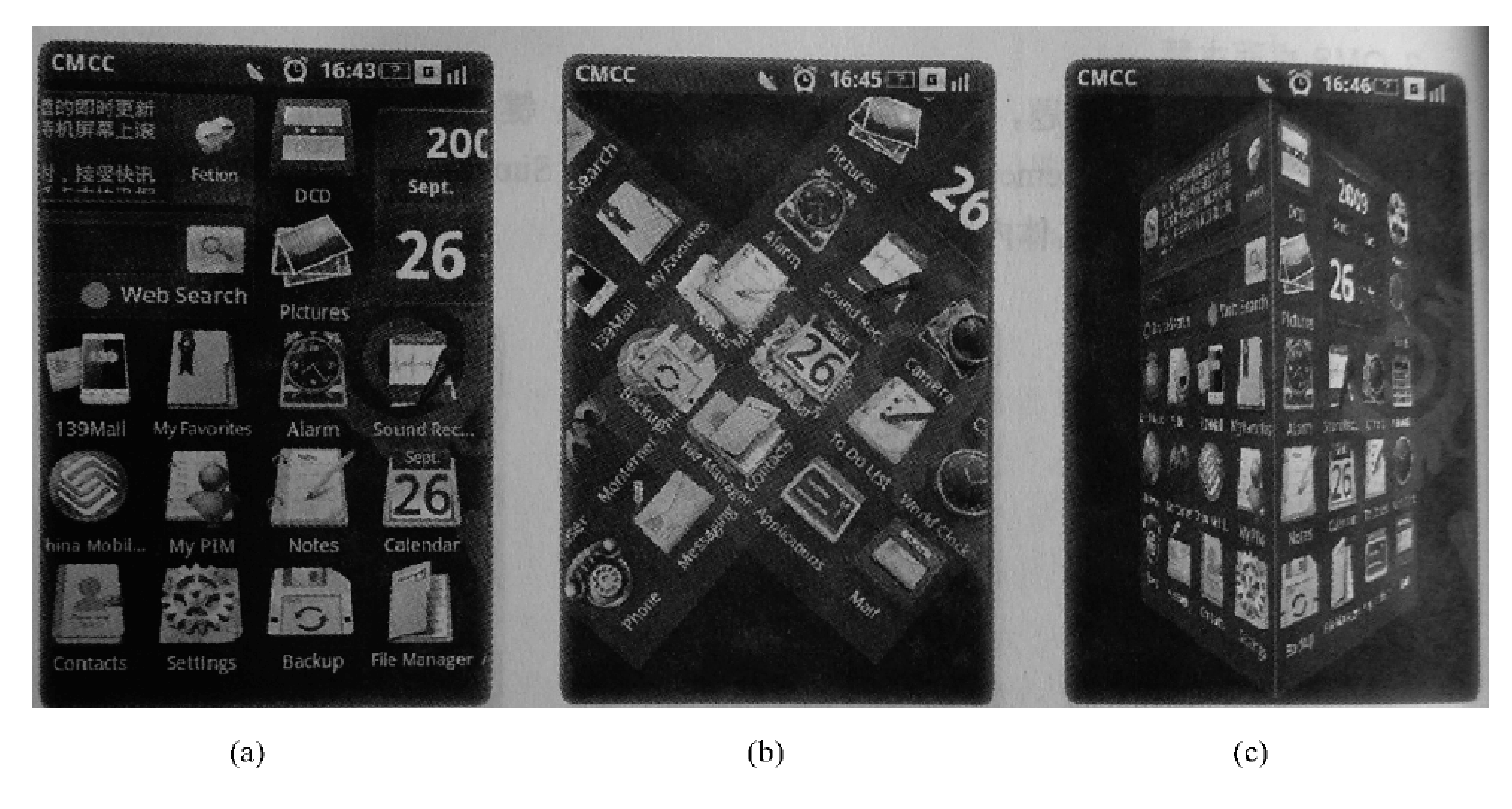
(a)　(b)　(c)

图 4-22　OMS 桌面主题

3）OMS 桌面动画风格

从 Android 上面只看到一种桌面动画风格，而 OMS 给人的第一视觉震撼就是它丰富而炫酷的桌面动画风格。从主页中选择 Menu 菜单，然后选择 Setting，可以看到 Screen Switch Effect，里面有桌面切换的动画风格，如图 4-23 所示。

(1) Flat(平滑动画)：切换桌面时，是水平方向左右平移的动画效果，如图 4-23(a) 所示。



(a) (b) (c)

图 4-23 OMS 桌面主题切换动画效果

(2) RotateCard(卡片动画):切换桌面时,以卡片形式显示,呈现出动画旋转的效果,类似于卡片翻滚效果,如图 4-23(b)所示。

(3) Cube(立方体动画):把桌面设置成多边立方体的一个面,切换时呈现出 3D 的动态效果,如图 4-23(c)所示。

4) OMS 桌面样式

除了类似于 iPhone 的桌面样式外,OMS 还提供了其他两种桌面样式。从桌面的 Setting 图标打开设置界面,在 Display Setting 中的 Home Setting 可以选择的有 3 种桌面样式:Tidy Home(精简主屏)、Florid Home(绚丽主屏)和 G1 Home(GI 主屏),分别如图 4-24 的(a)、(b)、(c)3 个图所示。



(a) (b) (c)

图 4-24 OMS 桌面样式

5）OMS 多桌面与桌面选择器

在 Android 中只有 3 个桌面可以使用，如果多加入几个 AppWidget 就没有空间来放入其他快捷方式与文件夹到桌面。而 OMS 可以设置多个桌面，在 Florid Home 绚丽主屏样式下，通过 Menu 菜单中的设置，打开 Screen Page Number 选项，就可以设置最大数为 9 的桌面主页。

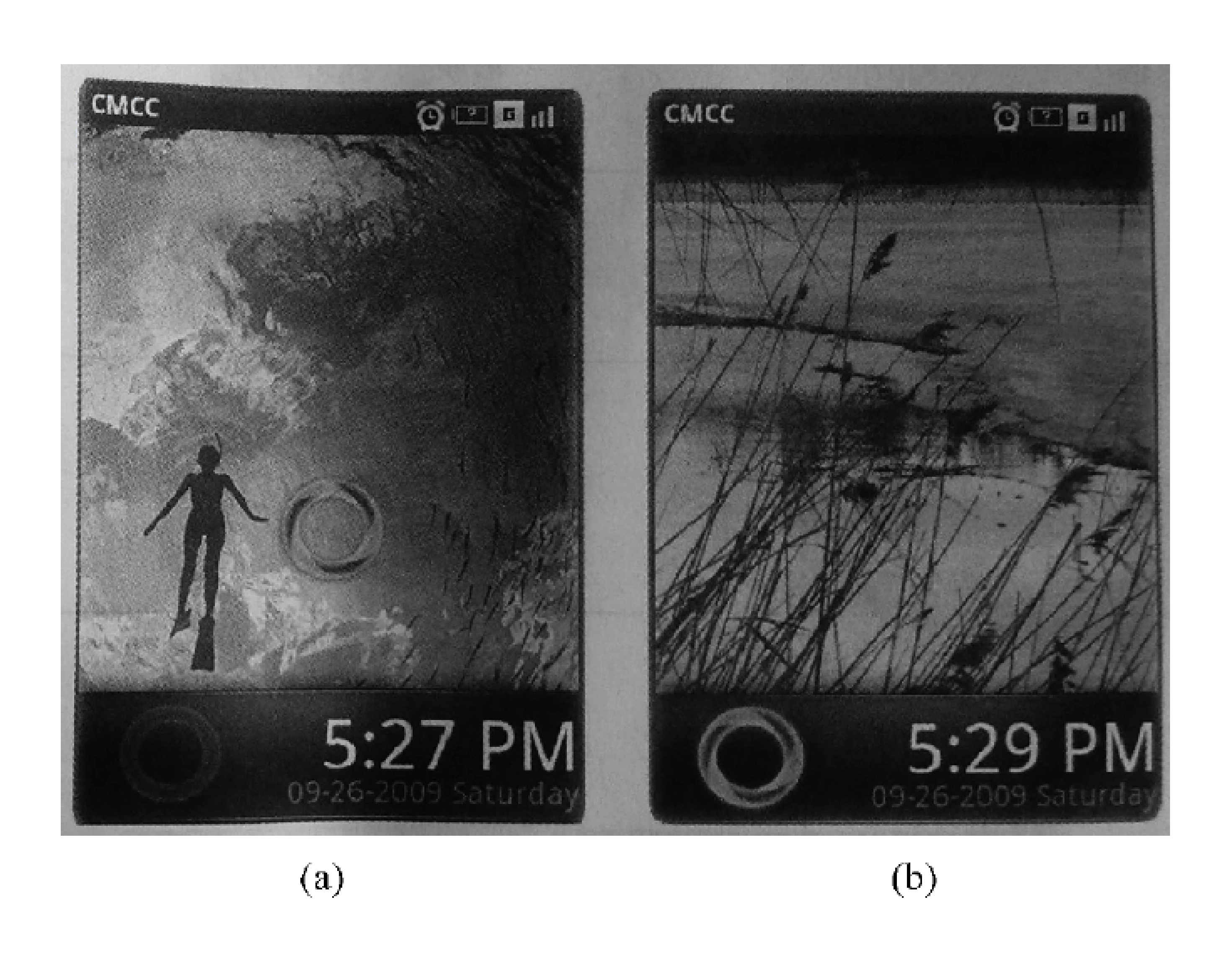


图 4-25　Home Switcher（桌面页数选择器）

OMS 提供了修改 Home 页面名称的功能，可以帮助用户在大量的 Home 页面中找到自己需要的页面，只要在页面最下方的 Home 字样按钮上长按，就会弹出页面名称修改的对话框，同时还提供了一个 Home Switcher，如图 4-25 所示，从而可以快速找到目标页面。

6）OMS 屏保

Android 的屏保提供了一个 Gesture 手势开锁程序，而 OMS 的开锁另有一番创意。OMS 进入屏幕保护以后会显示一个图片，这个图片可以自定义，桌面会有一个圆形的“O”标识，也就是 OMS 的标识，它进入浮游状态，如图 4-26(a)所示；开锁时单击它，然后拖动到指定的圆圈内即可解锁，如图 4-26(b)所示。



(a)　　　　(b)

图 4-26　屏幕锁定与解锁界面

7）OMS 的各种 UI 控件

OMS 中的 UI 控件名称及图示（或图标）如表 4-1 所示。

表 4-1　OMS 中的 UI 控件名称及图示(或图标)

名　　称	图　　标
AnalogClock	
AutoCompleteTextView	
Button	
CheckBox	
Chronometer	
DatePicker	
EditText	
ExpandableListView	
Gallery	
GridView	
ImageButton	

续表

名　称	图　标
ImageView	
ListView	
ProgressBar	
ProgressDialog	
RadioButton	
RatingBar	
SeckBar	
Spinner	

续表

名　　称	图　　标
ToggleButton	
Tab	
Timer Picker Dialog	
Data Picker Dialog	

从表 4-1 中可以看出，OMS 的 UI 控件大部分沿用了 Android 的样式，小部分采用了 OMS 独特的设计风格。在实际演示操作中会有更多的 UI 用户体验。

2. OMS 业务与自带的应用特色

OMS 的新业务和自带的应用与 Android 有很大区别，其特点如下：

(1) DCD 快讯。订阅快讯频道，接收各类频道的及时更新内容，以桌面 Widget 的形式动态滚动显示新闻以及其他资讯。

(2) 移动飞信。手机移动版本飞信业务，提供方便快捷的好友聊天(可多人)、发送文件、手机语音、直接呼叫已公开手机号码的好友。该飞信业务也提供了对一个或多个好友发送短信，集成了好友添加、删除、分组、本地联系人、黑名单功能，以及组管理、群管理的功能。让用户轻松地在手机上体验到和 PC 飞信一样强大的 OMS 飞信业务。

(3) 139 邮箱。让用户通过 OMS 手机终端阅读电子邮件正文和附件，回复、转发、撰写电子邮件。该移动邮箱集成了邮件阅读、发送、接收、搜索、账户管理等功能，让用户在高速、忙碌的生活中随时随地收发并阅读邮件。

(4) 无线音乐随身听。提供了方便的音乐随身听业务，以在线音乐与本地音乐功能的组合让用户随时随地都可以欣赏到优美动听的音乐。如果选择在线音乐 Tab，系统自动扫描在线音乐列表；如果选择了本地音乐 Tab，系统将自动扫描本地 SDCard 中的音乐文件列

表。系统也提供了以专辑、艺术家、歌曲名、风格的排列方式供用户选择,实现了随身听音乐的无穷乐趣。

(5) PIM 电话号簿管家。该软件以通讯录为基础,实现号簿更新,随时备份或恢复手机号簿和网络号簿的同步;提供了网络查询功能和同步服务,用户可把自己的电话簿放入网络存储,以便备份和更新;同时提供了网络同步、日志存储、日历、任务、数据连接等功能,让用户能够真正快捷、高效地管理自己的电话簿。

(6) 移动梦网。以 WAP 的形式提供梦网服务,让用户轻松体验移动梦网提供的优秀的移动服务。

(7) 中国移动服务。在此软件中提供了无线音乐、游戏社区、新闻天气、在线理财等移动服务,让用户随时随地享受中国移动的高质量服务。

(8) 待办事项。一个记事备忘录,记录需要完成事项的标题、开始时间、结束时间、任务详细内容、任务优先级和完成状态,即时提醒。

(9) 手机电视。提供了 CMMB 和 MBBMS 手机电视模式,用户可以在所在城市里扫描到电视节目;提供了在线扫描和本地电视播放,让用户享受到无线手机电视带来的乐趣;最有趣的是搭载了 OMS 的 OPhone,同时还提供一根外置天线。

(10) 本地、网络搜索。OMS 在桌面 Widget 上加入了一个新的搜索功能,让自己快速找到需要的应用与资源;提供了搜索存储卡和搜索内容类型功能;高级搜索提供了联系人、消息、电子邮件、文件、通话记录、浏览器等搜索项。搜索索引的建立以及索引分类功能这些人性化的搜索会让用户对 OMS 的优质服务体会得更加深刻。

OMS 通信功能、业务方面继承了优秀品牌的优良性,引入了诸多我国本土化的特色,主要表现在:

(1) 大屏幕全触摸的操作风格。

(2) 面向移动 Internet 应用的设计理念。

(3) 手写输入和拼音 T9 键盘的集成。

(4) 拼音和手写的切换。

(5) 拨号键盘可以用拼音直接调出联系人。

(6) 对话模式和文件夹模式可以随意选择的短信息用户界面。

(7) 彩信和短信结合的信息操作逻辑。

(8) 随意定制的主屏幕。

(9) 绚丽的动画。

(10) 奇妙的解锁方式。

3. OMS SDK 特色

在 OMS SDK 中,OMS 提供了一些和 Android 不一样的东西,下面从目录结构、其中的一些工具及特有的类库等方面来介绍 OMS SDK 的特色。

1) OMS SDK 目录结构

(1) dogs:存放了 OPhone 相关的介绍、实例、参考和 OMS Widget 的开发文档。dogs 分为两个部分,一部分是 OPhone 的参考文档,另一部分是 OMS Widget 的参考文档。

(2) samples:其中存放了 3 种 OMS 实例代码 Demo,即 Android APP 示例 Demo、OPhone APP 示例 Demo 和 OMS Widget 示例 Demo;6 个 Android 示例,即 ApiDemos、HelloActivity、

LunarLander、NotePad、SkeletonApp 和 Snake；3 个 OPhone 示例，即 HomeAPIDemo、MailAPIDemo 和 SearchAPIDemo；两个 OMS Widget 示例，即 APISample 和 HelloWidget。

(3) tools：存放了与 Android SDK tools 类似的开发工具，特有的一个 OPhone 目录中存放了 OMS 平台相关的 3 个工具，即 ADT-0.8.0.zip、jil-wdt-site.zip 和 ophone.sdk.doc_1.0.0.jar。其中 ADT 是 Eclipse 的开发插件，JIL-WDT 是开发 OMS Widget 的开发插件，ophone.sdk.doc_1.0.0 则是 oms.jar 的开发文档，其中包含 oms.Jar 包中类和接口的开发参考文档。

(4) Uninstaller：存放了 uninstaller.jar，即 OMS SDK 的可视化卸载器。一般双击或是进入该目录使用命令行执行 jave-jar uninstaller.jar 命令即可卸载。

(5) Installationinformation：OMS 的安装信息文档。

(6) Android.jar：从 Android 平台引用过来的 android.jar 包，包含 Android SDK 1.0 中的所有 Android API。

(7) Oms.jar：OMS 所特有的类库，含有 19 个包，其中的 location、mail、mms、mobilewidget、servo、sns 和 widget 包是 OMS 最具特色的类库包，可以用来开发 OMS 特色的应用程序。

(8) LICENCE_cn.txt：中文 OPhone SDK 版权许可协议。

(9) LICENCE_en.txt：英文 OPhone SDK 版权许可协议。

(10) RELEASR_TOTES.txt：OMS SDK 发布信息。

2) OMS SDK 独特类库

(1) oms.location：提供 GPS 定位、位置服务 API 和 GPRS 的位置服务。

(2) oms.mail：提供邮箱 CCMail、MIME 服务。

(3) oms.mms：提供移动短消息服务。

(4) oms.mobilewidget：提供移动访问控制、联系人信息、电话处理、SMS 服务等移动 Widget API。

(5) oms.servo.search：提供本地搜索服务。

(6) oms.sns：提供 Twitter 相关的 SNS 服务。

(7) oms.widget：提供 OMS UI、布局等视图 Widget。

3) OMS SDK 特有的 Demo

(1) HomeAPIDemo：主要利用 oms.home.HomeIntents 类实现 Home 中相应程序的 Symbol 和 Shortcut 添加或删除，展现了 oms.jar 包中新类库的使用。

(2) MailAPIDemo：向开发者展现了 oms.mail 包中 BaseAccountInfo、CMMail、CMMailAddress、CMMailTaskController、MailBaseEvent、MailEventListener、PSP3ProcessListener 的邮件核心类的使用与示例，让开发者更快地掌握 OMS 特有类库 oms.jar 的使用方法与核心类库的功能发挥。通过对 POP3 服务代理和端口设置以及 SMTP 服务代理设置，用户和开发者可以全新体验 OMS 带来的邮件服务功能。

(3) SearchAPIDemo：通过对关键字的设置，然后本地搜索所有相关的软件或资源，以文本的形式显示该搜索结果的 Resource ID、Title、Time、Mime 和 Size 值。典型运用到了 ome.servo.search 包里的 SearchProvider 类，做了全方位的示例，通过该 Demo 的学习可以在数据库、网络、搜索方面有所提高。

(4) APISample：这属于桌面 Widget 范畴。OMS 的桌面 Widget 采用了另外一种模式进行开发，即采用 JID-WDT 插件开发，Widget 应用采用 JIL(Joint Innovation Lab) Widget 标准，使用 HTML、JaveScript 和 CSS 等网络技术在 Widget 引擎上运行。通过对这个示例的学习可以体验到许多 JavaScript API 扩展 Widget 应用的能力以及开发 OMS Widget 的高效快捷。

(5) HelloWidget：从这个示例中可以了解与掌握基础 OMS Widget 开发的过程与知识。该程序的主要作用是在主屏上显示一个 OMS 特有的 Widget 样式，显示“Hello Widget!!”字样。

4) OMS Emulator 与版本

OMS 的 Emulator 也富有特色。相比 Android，OMS SDK 中提供了另外两个新增加的 Emulator 皮肤 Benzina 和 HVGA；在这两个典型的皮肤中两个 Emulator 皮肤对应的 Emulator 内容也是不同的，Benzina 对应于已经发布的 Dell 实机，HVGA 对应于联想发布的 OPhone 版本。目前已经有很多厂家发布了基于 OMS 操作系统的 OPhone 系列手机，例如戴尔的 Dell Mini 3i、摩托罗拉的 MT710、联想 O1、多普达的 A6188、飞利浦 V900 和海信 E3 等。

4. OMS 和 Android 的关系

OMS 基于 Linux 内核，采用大部分 Google Android 源码，是经过定制 Google Android 平台演变出的一种更适合我国用户的智能手机操作系统，在 Google 的 Android 系统之上二次开发而来。从技术角度来讲，OMS 属于 Google Android 的定制版，它分为底、中、表三层，底层就是 Linux 内核层面，中间层则是一个叫做 Dalvik 的 Java 虚拟机，表面层则是根据 Android 系统修改定制而来的运行库。每个应用程序都运行在自己的进程上，享有 Dalvik 虚拟机为其分配的专有实例。OMS 系统与 Android 系统的应用软件都是运行在 Dalvik 之上的 Java 软件。OMS 可以完全兼容 Android 1.1 的所有应用，部分兼容 Android 1.5 版本后的应用。OMS 在 Android 的基础上又加入了自己特有的一套 OMS 类库，加入了更多的中国移动业务。Android 1.1 之前的程序可以完全在 OMS 上正常运行，但是 OMS 的应用不一定在 Android 上运行，因为 OMS 独特的类库，使得 OMS 的程序在一定程度上放到 Android 上面运行需要进行移植工作，主要是运行相似的 API 对 OMS 的程序做移植。图 4-27 所示为 OMS 与 Android 的关系。

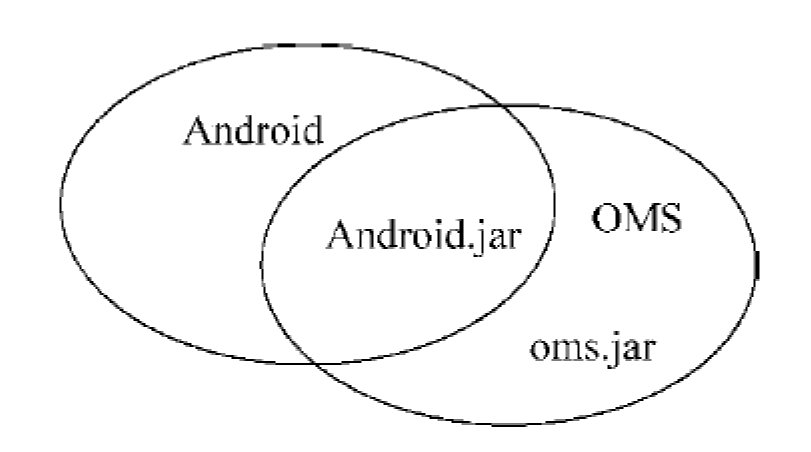


图 4-27　OMS 与 Android 的关系图

4.4.3　普通 Android 应用向 OMS 平台迁移

1. OMS 开发平台搭建

如果对 J2ME、Android、Symbian、BlackBerry 等智能手机开发非常熟悉，就会知道其开发平台的搭建具有共同的地方。首先安装 JDK，然后安装开发插件到 Eclipse，最后指定 SDK 开发包的位置。平台搭建好后，OMS 搭建的开发环境和 Android 非常类似，OMS 加入了更多的移动服务 API、另类的 Widget 开发框架，在开发过程中需要开发相应的中国移动的一些移动服务应用，需要导入 oms.jar 第三方 jar 包，导入 OMS API。对于 Widget 的

开发模式，与 Android 1.5 之后的 WidgetFrameWork 也有着天壤之别。下面简单介绍 OMS 开发环境的搭建过程。

1）软件下载

（1）进入 http://www.xin126.cn/soft_show.asp?id=17，下载 JDK1.6，如图 4-28 所示。



图 4-28　JDK1.6 下载

（2）进入 eclipse.org，下载 Eclipse IDE for Java Developers 版本的 Eclipse，如图 4-29 所示。

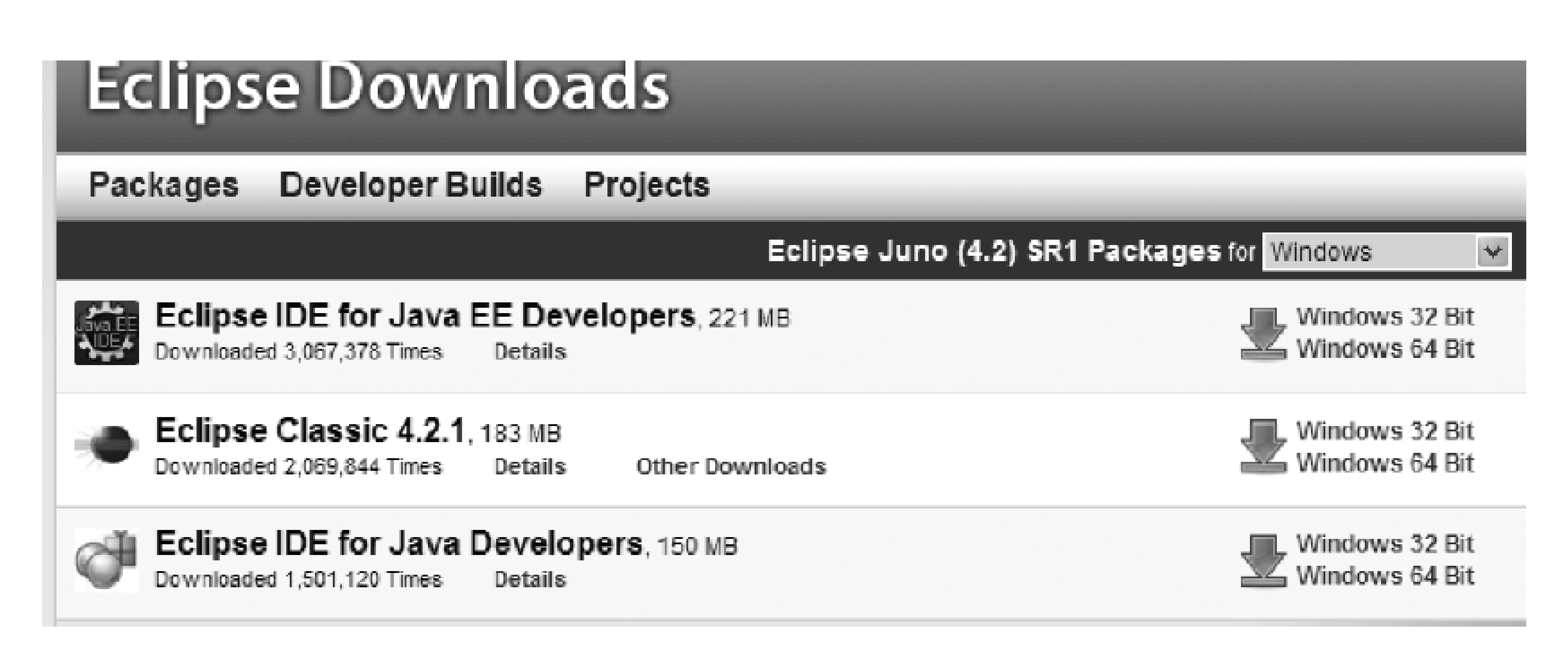


图 4-29　Eclipse 官方 Eclipse Ganymede for Java 下载

（3）进入 OMS SDN 开发网 ophonesdn.com，从该网站下载 SDK，单击“立即下载”按钮后，弹出注册界面，如图 4-30 所示，提示需要先注册才能下载，其下载界面如图 4-31 所示。

2）安装 JDK

安装 JDK 的方法与普通程序的安装一样，在此不再叙述。安装好后，在环境变量中加入以下环境配置：

图 4-30　Ophone SDN 注册和登录界面

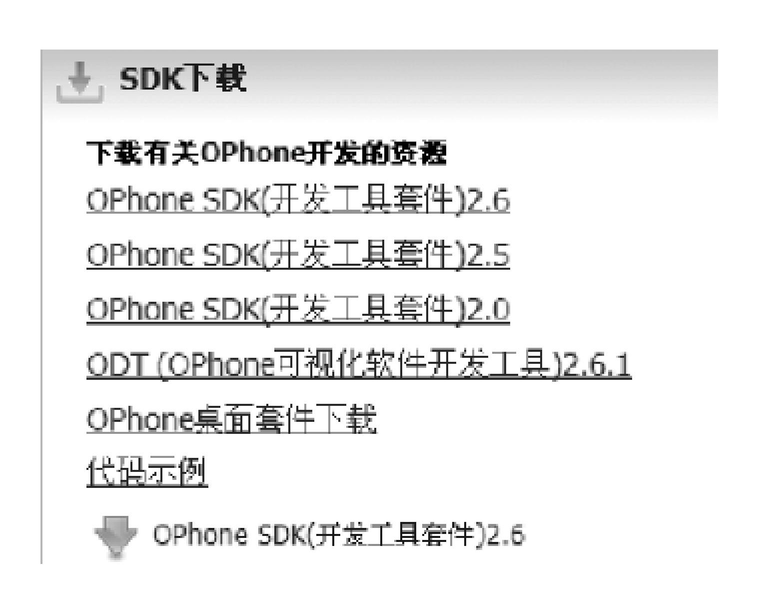


图 4-31　OMS SDN 官网 OMS-SDK 下载

```
CLASSPATH
.;C:\Program Files\Java\jdk1.6.0_16\lib;C:\Program Files\Java\jdk1.6.0_16\lib\dt.jar;
C:\Program Files\Java\jdk1.6.0_16\lib\tools.jar;
C:\Program Files\Java\ jdk1.6.0_16\jre\lib;
C:\ProgramFiles\Java\jdk1.6.0_16\jre\lib\rt.jar;
C:\Program Files\Java\jre6\lib;
C:\Program Files\Java\jre6\lib\rt.jar
PATH
C:\ProgramFiles\Java\jdk1.6.0_16\bin;
C:\Program Files\Java\jdk1.6.0_16\jre\bin;C:\Program Files\Java\jre6\bin;
```

```
 %SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;
C:\Program Files\ESTsoft\ALZip\;C:\Program Files\TortoiseSVN\bin;
C:\Program Files\Common Files\Thunder Network\KanKan\Codecs
```

3）安装 Eclipse

把刚刚下载的 elipse-java-ganymede-SR2-win32.zip 解压到 E:\，解压后的目录为 E:\Eclipse。

4）安装 OMS SDK

（1）把刚刚下载的 ophone-sdk_windows-1.0-setup.jar 复制到 E:\。

（2）现在安装 OMS_SDK，安装到 E:\OMS-SDK 目录中有以下两种方法。

方法一：双击。由于 OMS_SDK 的开发包已经制作成一个 Jar 包，可以像安装普通软件一样双击安装。

方法二：使用 CMD 进入 E:\，运行以下命令：

```
Java-jar ophone-sdk_windows-1.0-setup.jar
```

5）安装 ADT

启动 Eclipse，然后安装 ADT0.8.0Android 开发插件，具体方法如下：

（1）选择 Help→Software Updates，如图 4-32 所示。

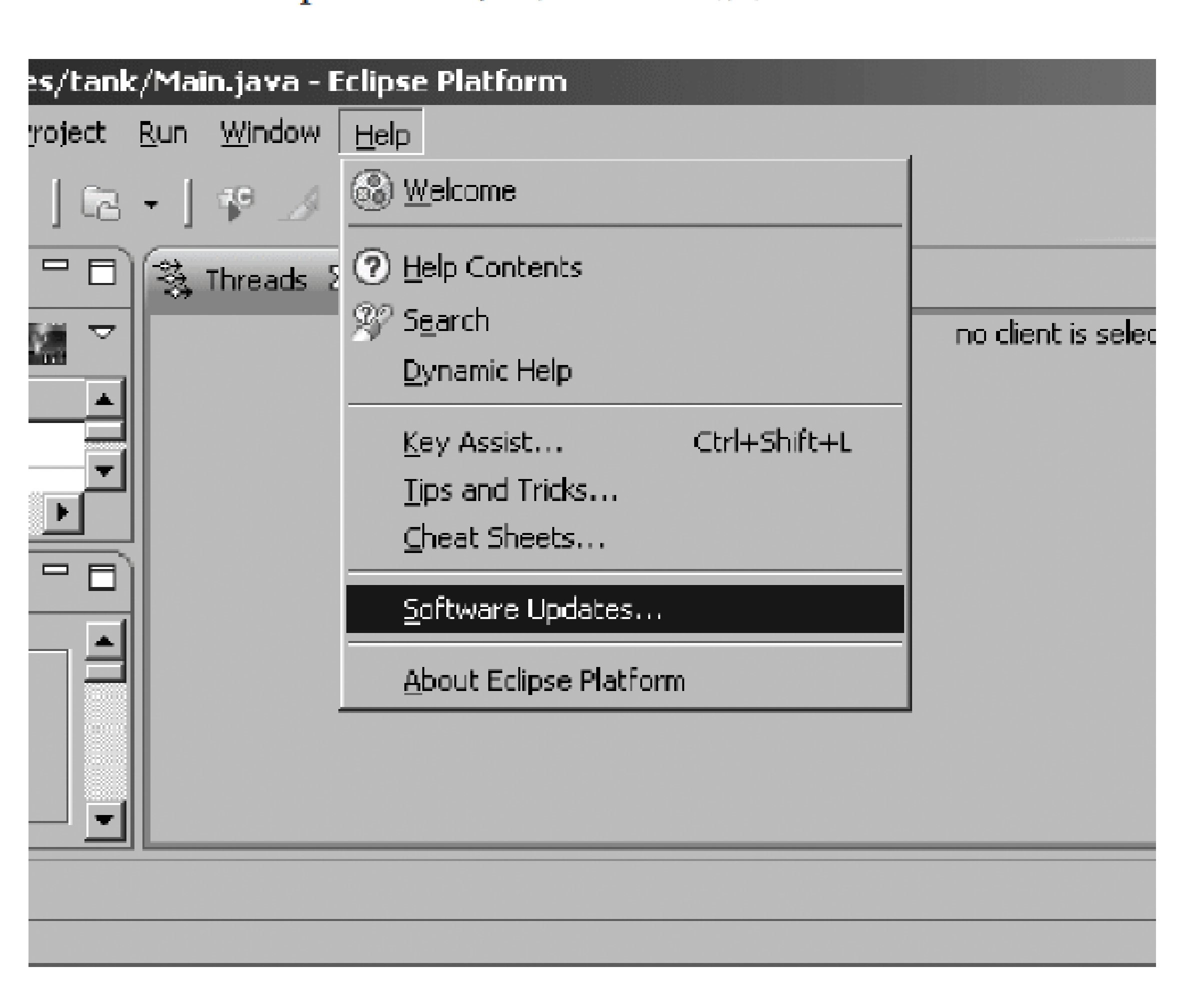


图 4-32　选择 Software Updates 安装 Ecilipse 插件

（2）在 Software Updates and Additions 对话框中单击 Available Software 标签下的 Add Sites 按钮，如图 4-33 所示。

（3）在 Add Site 对话框中单击 Archive 按钮，如图 4-34 所示。

（4）在弹出的 Repository archive（选择归档）对话框中，进入到 E:\OMS-SDK\tools\ophone 目录中选择文件 ADT-0.8.0，如图 4-35 所示。

（5）ADT 安装完毕后还要简单配置一下，打开菜单中的 Window→Preferences，如图 4-36 所示。

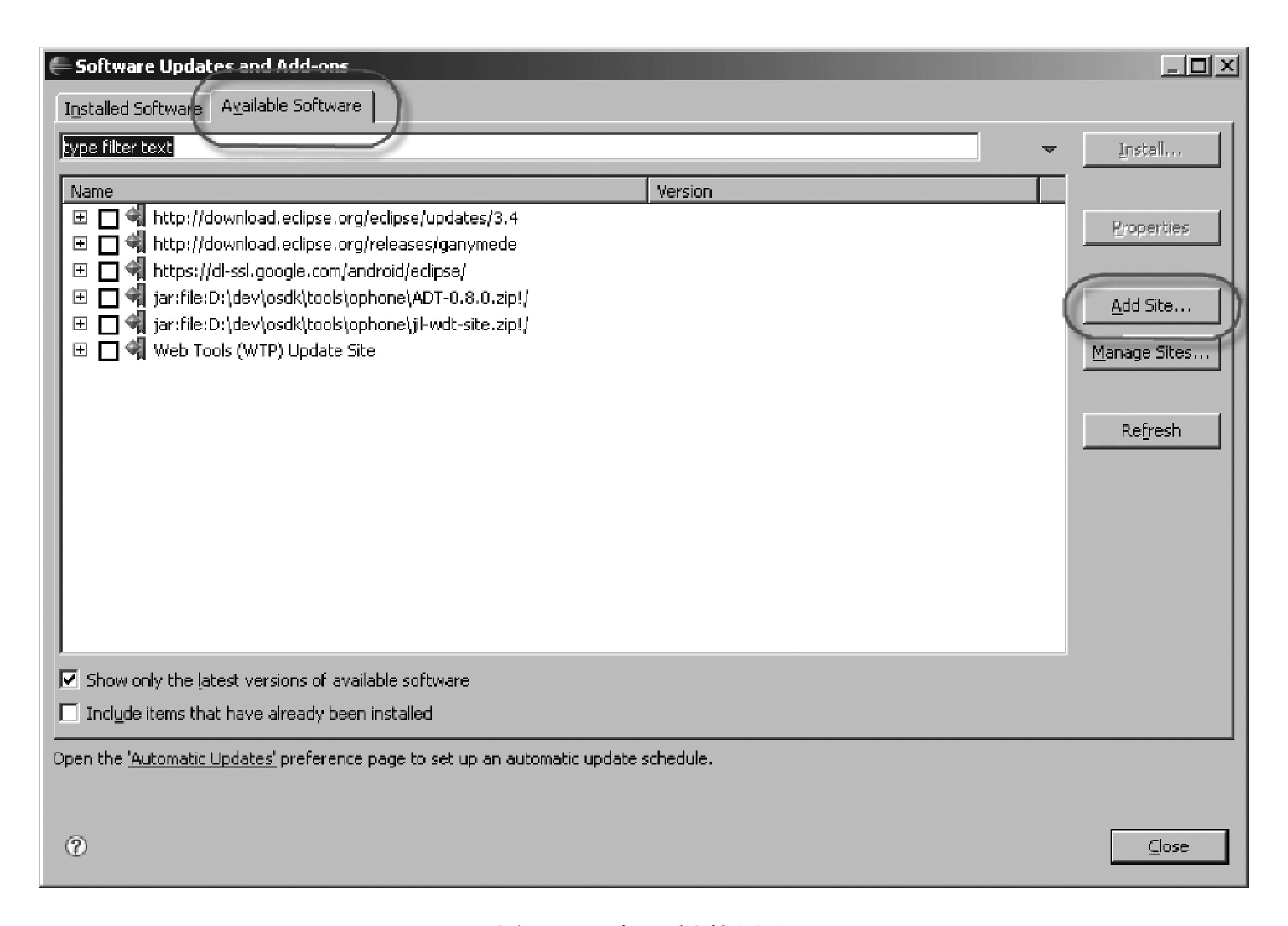


图 4-33　加入插件网址



图 4-34　使用本地插件归档

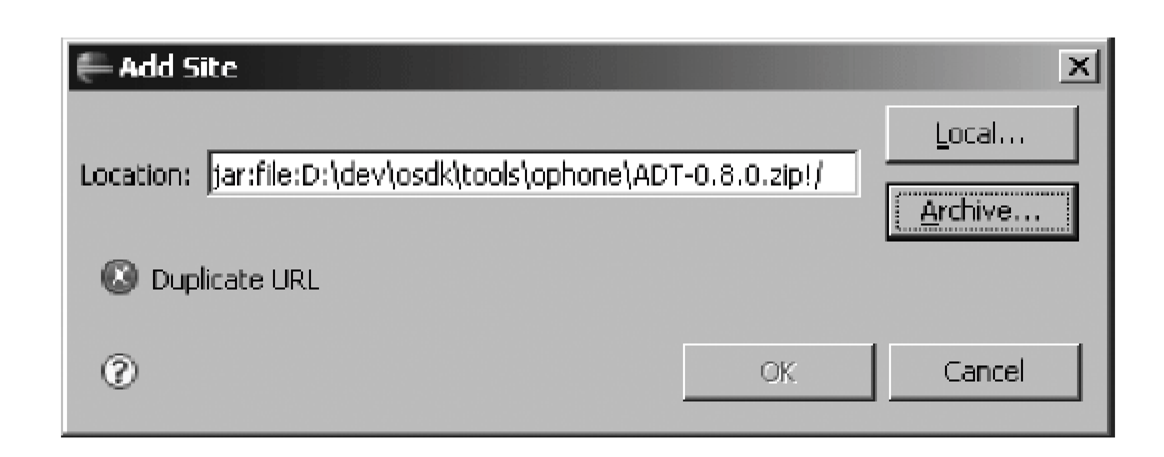


图 4-35　选中 ADT-0.8.0 插件本地归档

(6) 找到 Android 选项，通过 Browse 按钮指定 Android SDK 的安装位置，如图 4-37 所示。

至此，Android 的安装环境就全部搭建完毕了。

(7) 进入协议确定对话框，选择同意，然后单击 Finish 按钮，如图 4-38 所示。安装完成后重启 Eclipse。

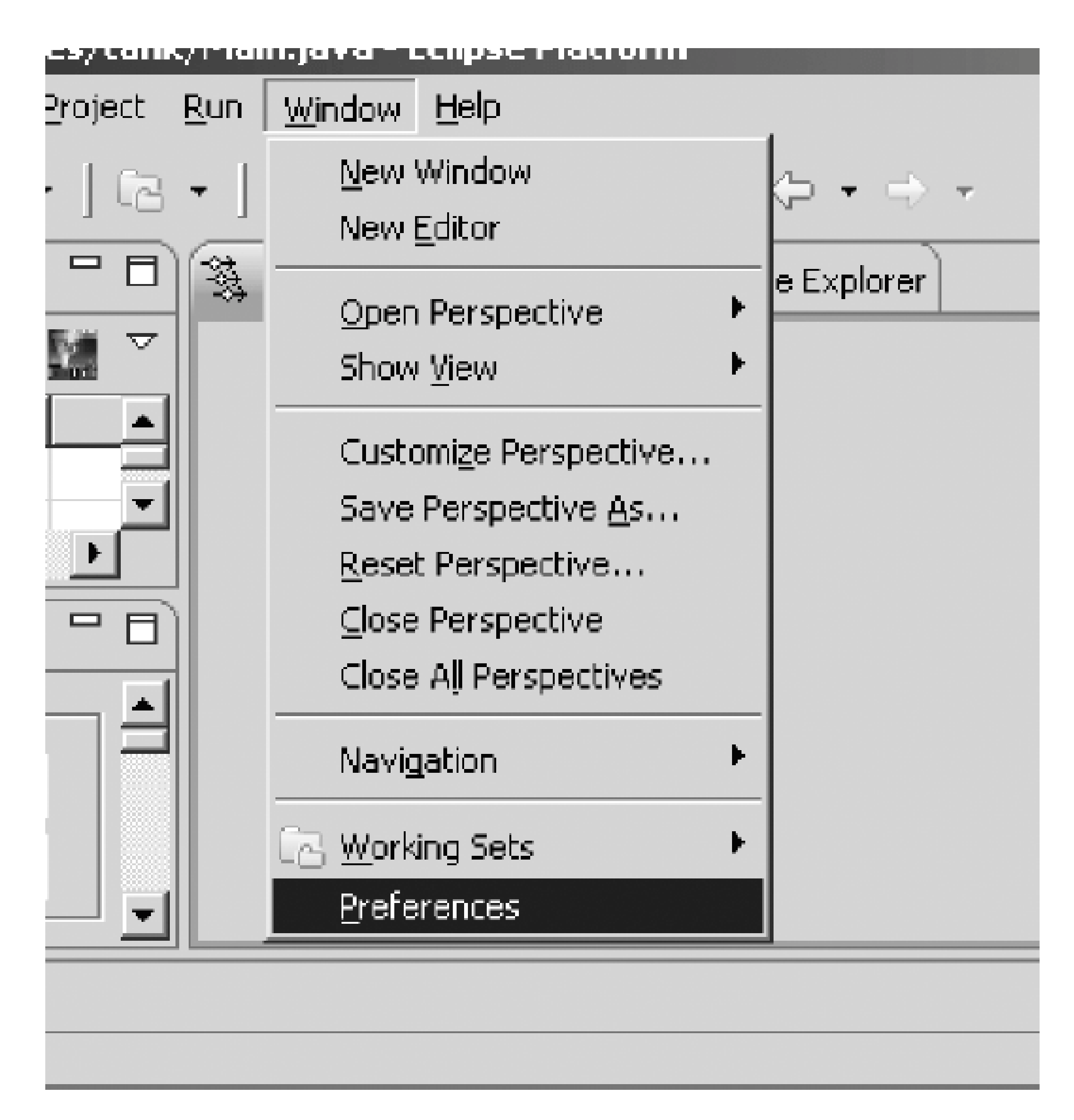


图 4-36 配置 ADT 插件

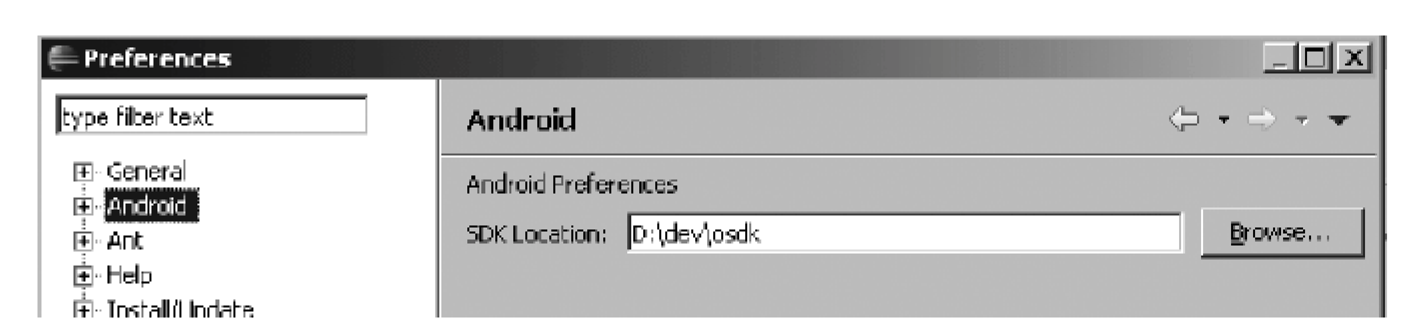


图 4-37 指定 Android SDK 的安装位置

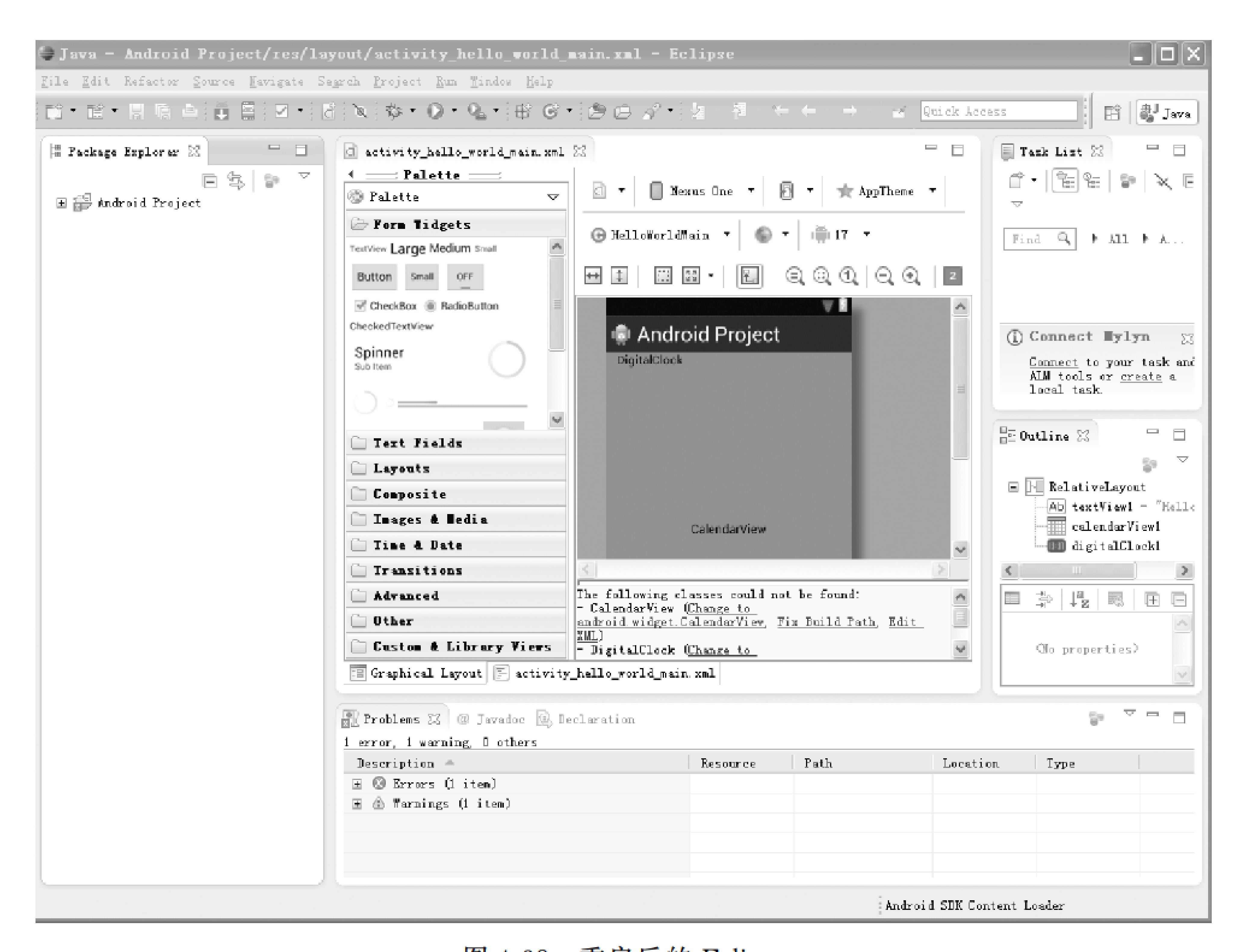


图 4-38 重启后的 Eclipse

6）配置 OMS-SDK 路径

（1）选择 Window→Preferences，如图 4-39 所示。

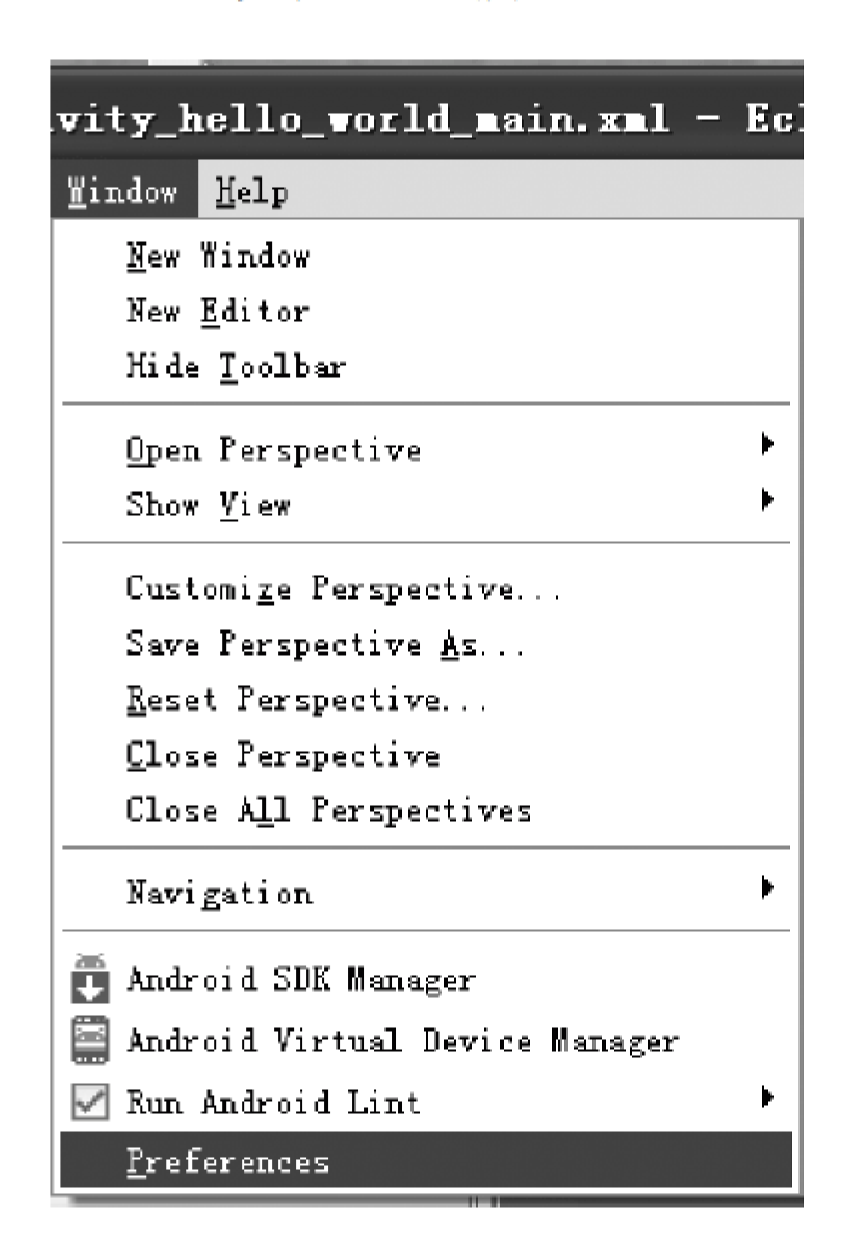


图 4-39　选择 Window Preferences 菜单项

（2）在 Preferences 对话框的左侧选中 Android 目录项，如图 4-40 所示。

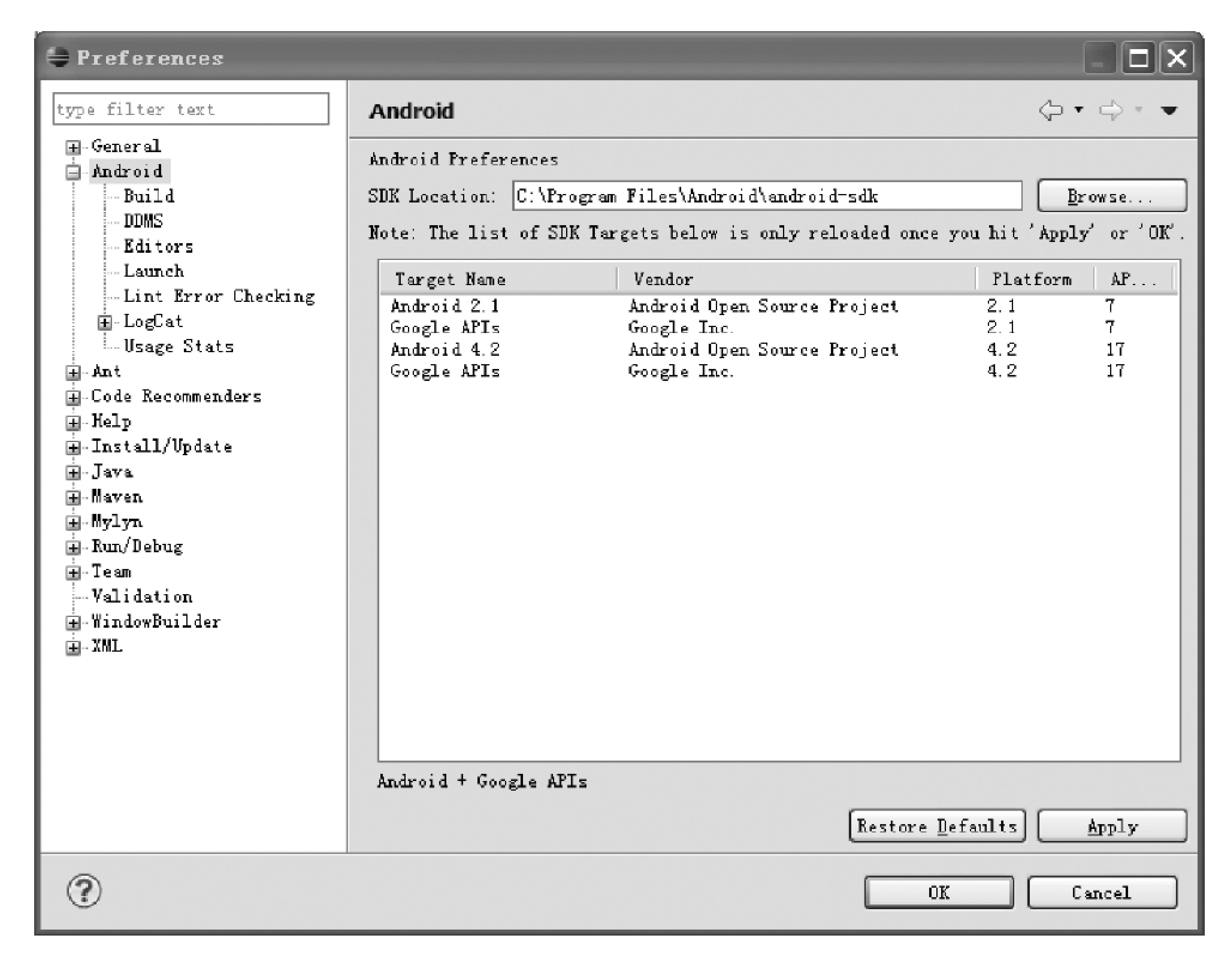


图 4-40　选择 Android 目录项

(3)在如图 4-41 所示的对话框中选中 OMS-SDK 根目录,单击“确定”按钮。



图 4-41　指定 OMS SDK 位置

7) 加载 OMS USE Library

在 E:\OMS-SDK 目录下,也就是 OMS-SDK 根目录中,除了 android.jar 以外,OMS 还提供了 oms.jar,其内部包含了中国移动的一些移动服务接口类 API。需要开发基于 OMS 的中国移动服务应用时,就可导入 oms.Jar 包。

(1) 选中 Window→Preferences,在 Preferences 对话框左边选择 Java 目录下 Build Path 中的 User Libraries,如图 4-42 所示。然后单击 New 按钮。

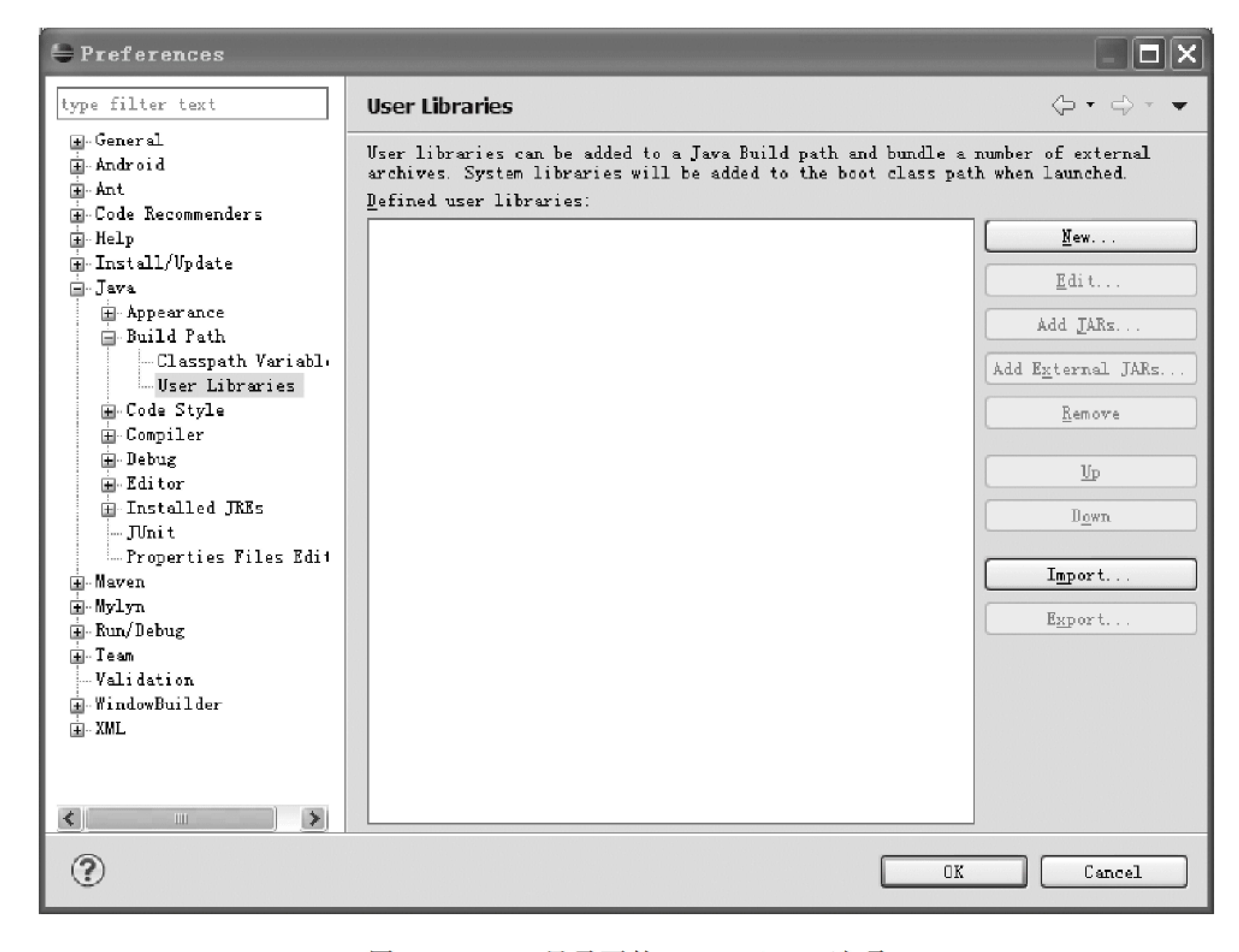


图 4-42　Java 目录下的 User Library 选项

(2) 在 New User Library 对话框中输入“OMS”,并选中 System Library 复选框,然后单击 OK 按钮,如图 4-43 所示。

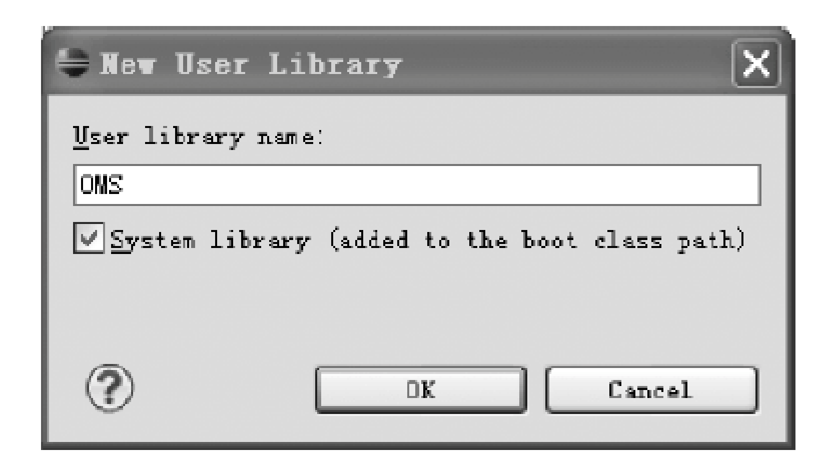


图 4-43　新加入 OMS User Library

(3) 选择 User Librares 右边的 Add JARs 按钮,如图 4-44 所示。

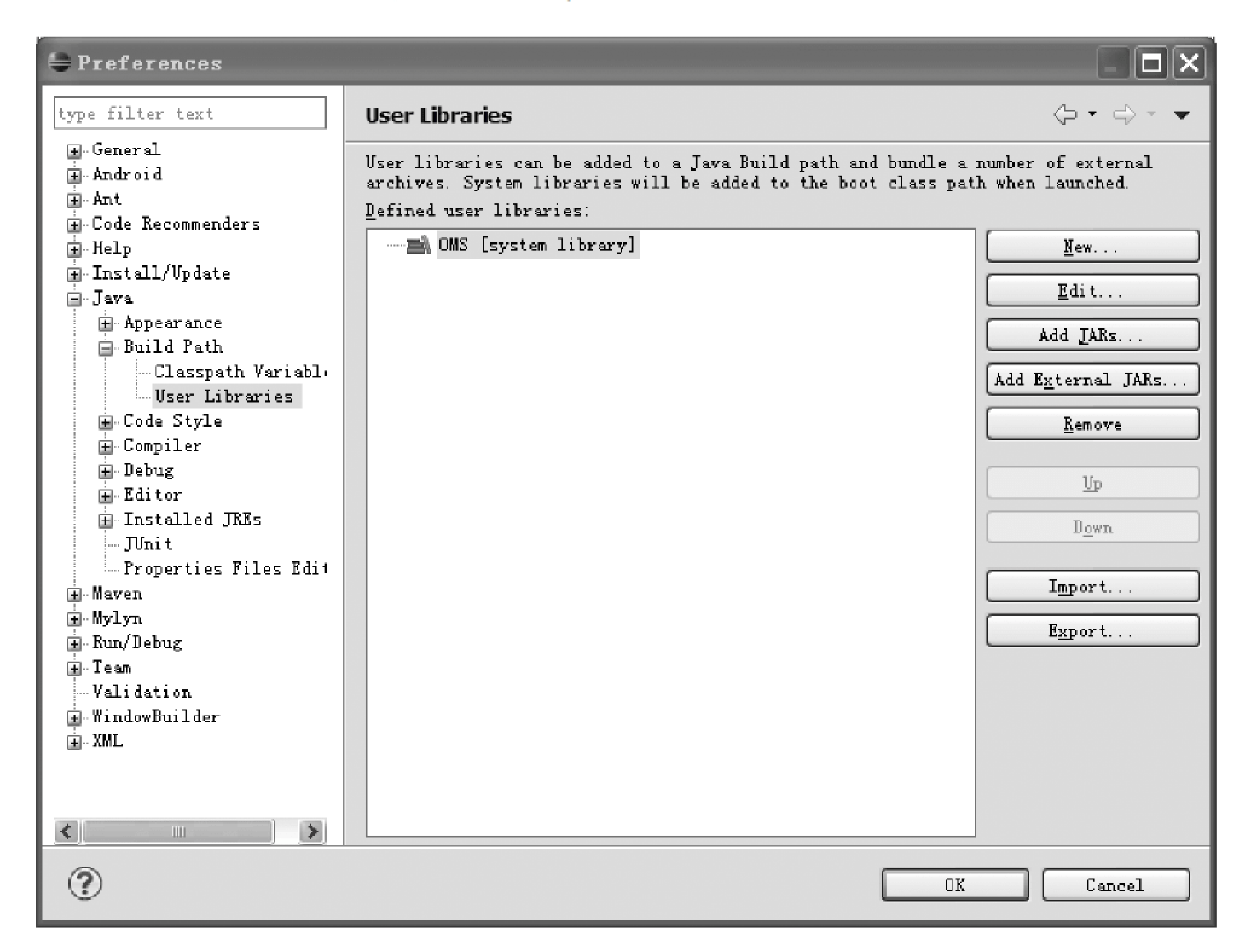


图 4-44　添加 Jar 包

(4) 在弹出的 JAR Selection 对话框中,选择 E:\OMS-SDK 目录中的 oms. jar,然后单击“打开”按钮,如图 4-45 所示。

(5) 在 Preferences 对话框中的 User Libraries 区域展开刚刚加入的 oms. jar,选中 Javadoc location,然后双击它或单击右边的 Edit 按钮,如图 4-46 所示。

(6) 在 Javadoc For oms. jar 对话框中选中 Javadoc in archive 单选按钮,然后选中 External File,最后在 Browser 中选中 E:\OMS-SDK\tools\ophone 目录下的 ophone. sdk. doc_1. 0. 0. jar,如图 4-47 所示。单击 OK 按钮后完成配置。

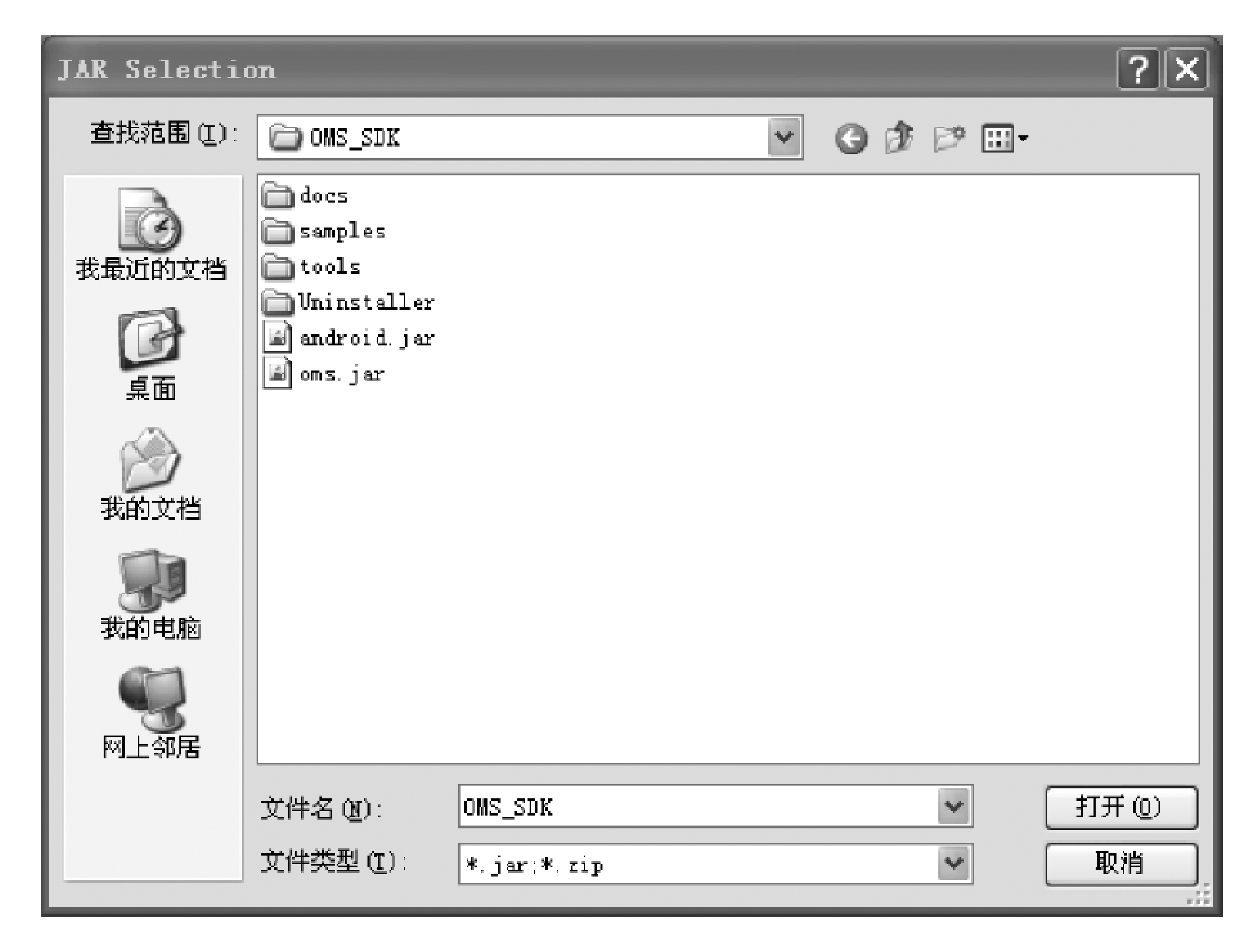


图 4-45　指定 OMS Jar 包位置

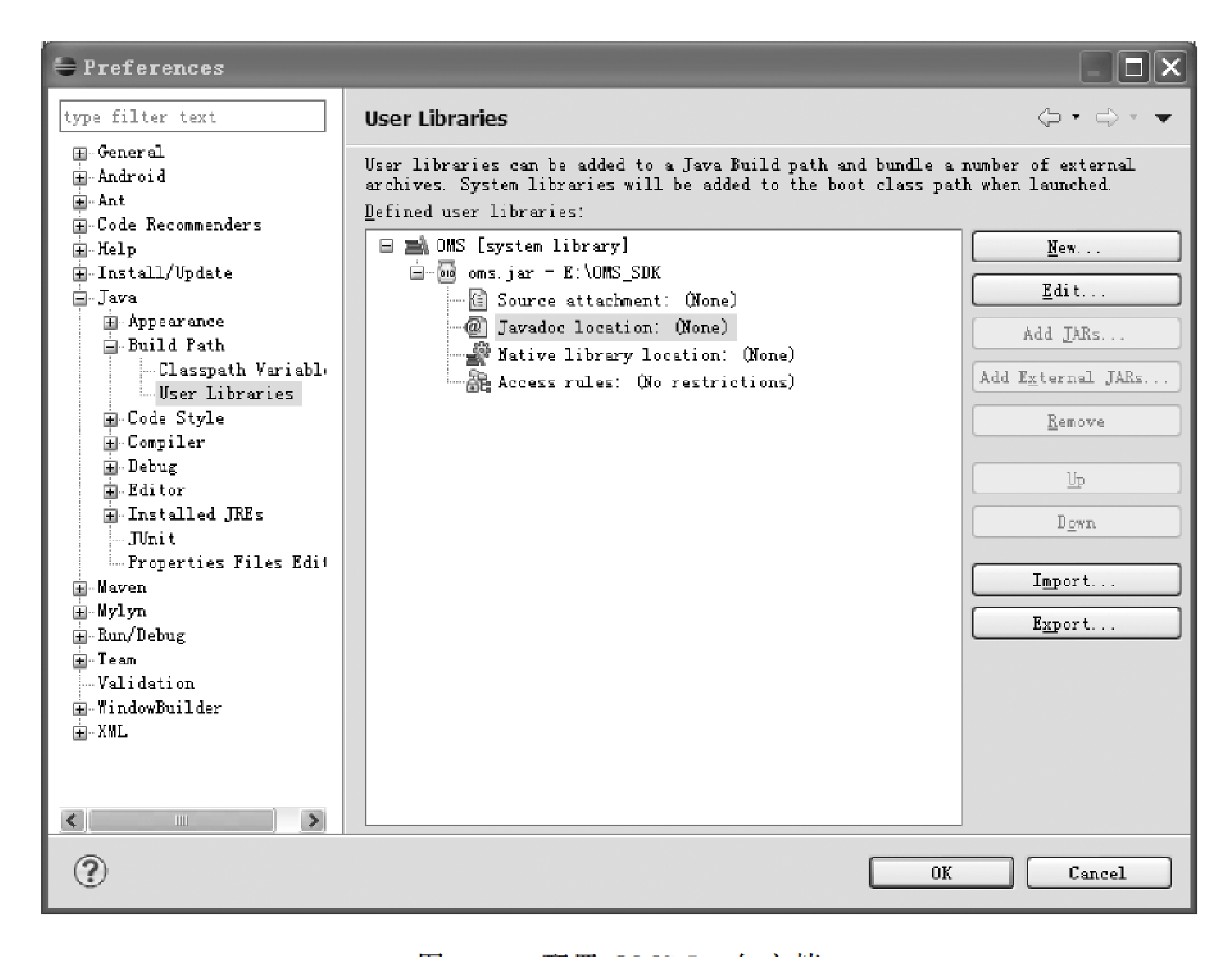


图 4-46　配置 OMS Jar 包文档

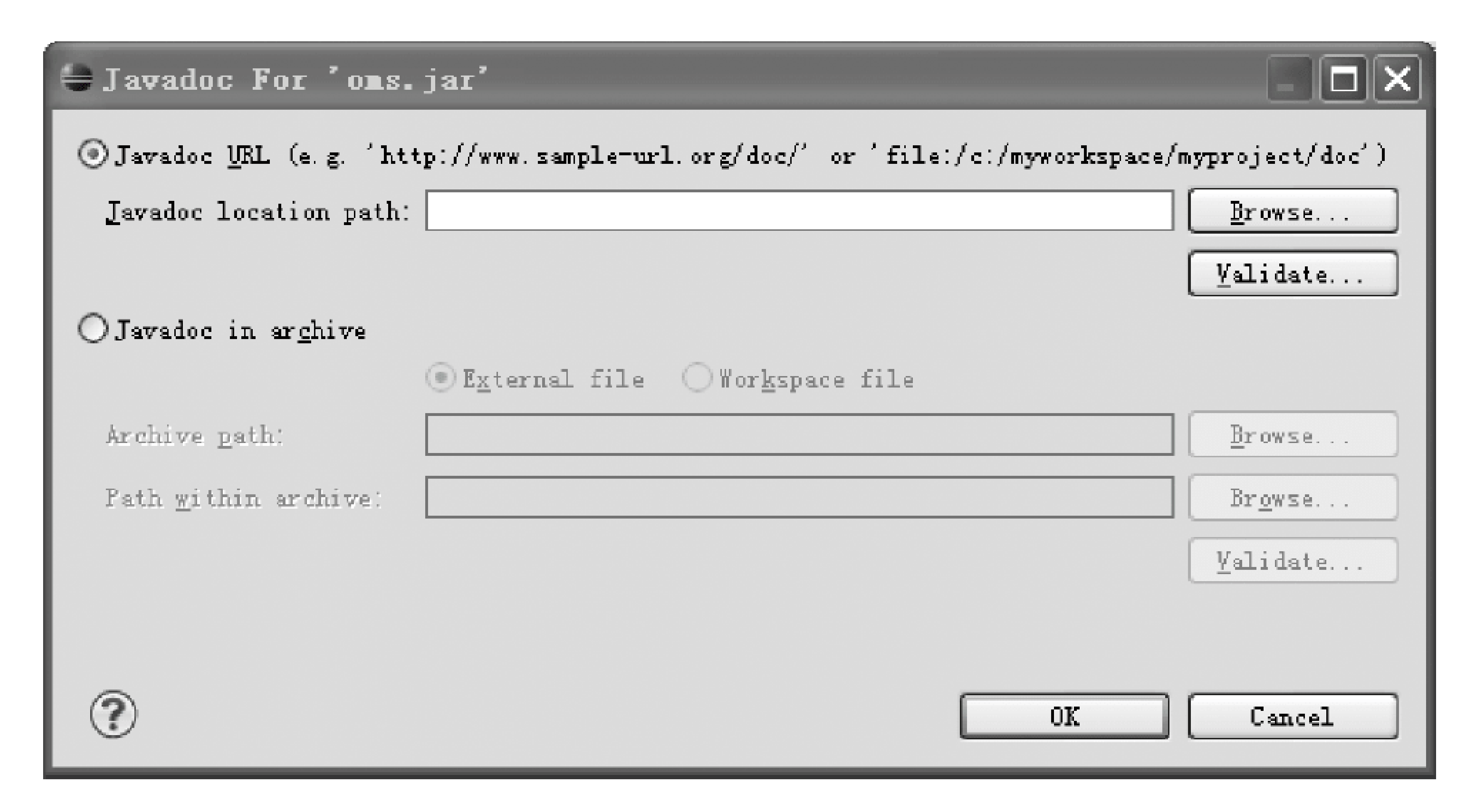


图 4-47 指定 OMS Jar 包文档本地归档路径

2. OMS 应用和 Android 应用的差异

Android 的程序不能完全在 OMS 上兼容，OMS 的部分应用在 Android 上无法运行。这主要是由于 OMS 应用和 Android 应用存在以下两点差异。

1) API 类库差异

Android 采用的是 android.jar 类库，其中还有 Android 平台提供给开发者的一些开发 API 类库。按照版本的升级，有一部分类库会被丢弃或是使用新的类库来替代，实现更新升级。目前，Android 版本已经升级到 2.0，Android SDK 版本已经升级到 1.6r1 版本，与 2009 年 5 月发布的 Android SDK 1.1r1 相比，一部分类库与 API 也发生了巨大的变化，加入了很多新鲜的元素，但不断的升级过程会造成部分类库的不兼容问题。OMS 采用的是 Android 1.0 类库，与 Android 相比相对落后，目前 OMS SDK 正式上线升级为 Ophone SDK 1.0，它的类库中，也就是 android.jar 以及 SDK 目录节后，相对于 Android 1.5 来说，结构完全不一样，而且自 Android 1.5 后，Android 加入了 AppWidgetFrameWork 框架和 AVD，这些在 OMS 上是没有的，OMS 的 Widget 也采用了另一种 JIL 的开发模式进行开发。所以，Android1.1 以及以前的应用在 OMS 上可以完全兼容，Android 1.5 以后的应用在 OMS 上部分兼容；OMS 在 Android 基础上加入了自己的业务和类库，也就是 oms.jar 以及中国移动的一些移动服务。如果使用了 oms.jar 类库中的类，编写出的应用在 Android 中会出现不兼容问题。

2) Android 版本差异

目前，Android 升级到 2.0，Android SDK 还是 1.6r1 版本，而 OMS 之前一直都是采用 Android 1.0 或是 Android 1.1 的大部分源码和框架，这样随着两个版本的升级、修改与定制，导致了版本不一致的兼容问题，最后导致结构、框架和 API 类库的不兼容。

3. 普通 Android 应用向 OMS 平台迁移

1) 解决不兼容的问题

Android 1.5 之后的应用迁移到 OMS 平台类似于向低版本的 Android 1.1 之前的版本

迁移,通常采取以下方法处理这个问题。

(1) 确定 Android 程序所使用的 Android SDK 开发版本,如果它是 Android 1.1 或是之前版本,不需要修改代码,即可把 Android 应用放到 OMS 上运行;如果它是 Android 1.5 或是之后的版本,就需要进行更多的代码移植工作。

(2) 检查自己的类库,确定需要迁移的应用所使用的类库,对比与 OMS 上的差异,然后计划出一个合适的方案,如果在 OMS 上有该类和一些可用方法,可直接使用 OMS 上提供的类和方法。如果该类或方法在 OMS 上面没有出现过或是 Android 1.1 之前都存在的方法,而到了 Android 1.5 已经丢弃了,或是用新的方法代替了,则需要使用旧的 API 和类库进行代码修改,以及类功能的替换。

(3) 在完成类库与 API 的替换修改后,我们要做的就是把 Android 1.5 以及 Android 1.5 以后版本 SDK 开发的应用项目中的 Android Manifest.xml 配置文件中的

```
<uses-sdk android:minSdkVersion = "3"/>
```

或是

```
<uses-sdk android:minSdkVersion = "4"/>
```

代码删除,因为该属性是 Android1.5 以及之后版本特有的属性,用来制定 API 等级。

(4) 完成以上三个步骤后,需要放到 OMS 上运行一下,看看还有哪些问题。一般就是屏幕的适配问题,因为 Android 的屏幕配置一般为 320×480 像素,而 OMS 的屏幕不定,例如 DELL 的 MINI 3i,该手机搭载了 OMS,屏幕像素就是 640×360,需要注意迁移后的布局失真;接下来是定义并修改布局和 UI 方面,例如文字的字体、颜色、大小等,需要适配新的环境以完善效果。

需要注意的是,Android 1.5 之后的 APPWidget 无法迁移到 OMS 上,因为这两个开发环境中的桌面 Widget 编程方式不同,Android 1.5 采用的是 APPWidget FrameWork,而 OMS 采用的是 JIL,是利用 HTML、CSS、JavaScript 等网络技术来实现的。

2) 开发中常遇到的两个问题

(1) 在 Android 1.5 上利用 SoundPool 编程实现对 MIDI 音乐文件的播放,而移植到 OMS 后发现 OMS 在 SoundPool 和 MIDI 音乐文件的支持方面还不完善,会出现机械的轰鸣噪声,非常刺耳。为了解决这一问题,需要对以前的 MIDI 音乐文件转换格式,一般转换成 OGG 格式,但转换后的文件会明显增大,导致程序的臃肿。

(2) 在 Android 1.5 上使用 Sensor 开发出的重力感应程序,移植到 OMS 时发现,在 OMS 的 android.jar 包的 android.hardware 包中只看到了 SensorListener 接口和 SensorManager 类,而在 Android1.5SDK 中提供了两个接口 SensorEventListener 和 SensorListener,以及三个类 Sensor、SensorEvent、SensorManager。在 Android 1.5SDK 开发的重力感应程序中使用的类和接口在 OMS 上无法使用。因此,需要在 OMS 版本上采用 Android 1.1 之前的 Sensor 编程方式类修改代码。

第5章 Android 应用程序

本章全面介绍 Android 应用程序开发的前期工作，并介绍 Android 游戏编程实例(Tank 大战)，为读者在以后的 Android 应用程序开发中奠定基础。通过本章学习，读者应该掌握以下内容：

(1) 搭建开发环境。

(2) Android 应用程序的结构。

(3) Android 的虚拟机和 Java 环境。

(4) Android 用户界面开发。

(5) Android 游戏编程——Tank 大战。

5.1 搭建开发环境

Android 是由 Google 开发的、基于 Linux 内核的软件平台和操作系统。自它诞生以来，它的功能和易用性不断得到完善。本节内容适合零基础的开发人员。

5.1.1 Windows 7 下 Android 开发环境搭建

1. JDK 安装

1) 下载 JDK

(1) 如果还没有 JDK 的话，可以进入页面 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下载最新版本的 JDK，即 Java SE 6 Update 31，详细的下载页面如图 5-1 所示。

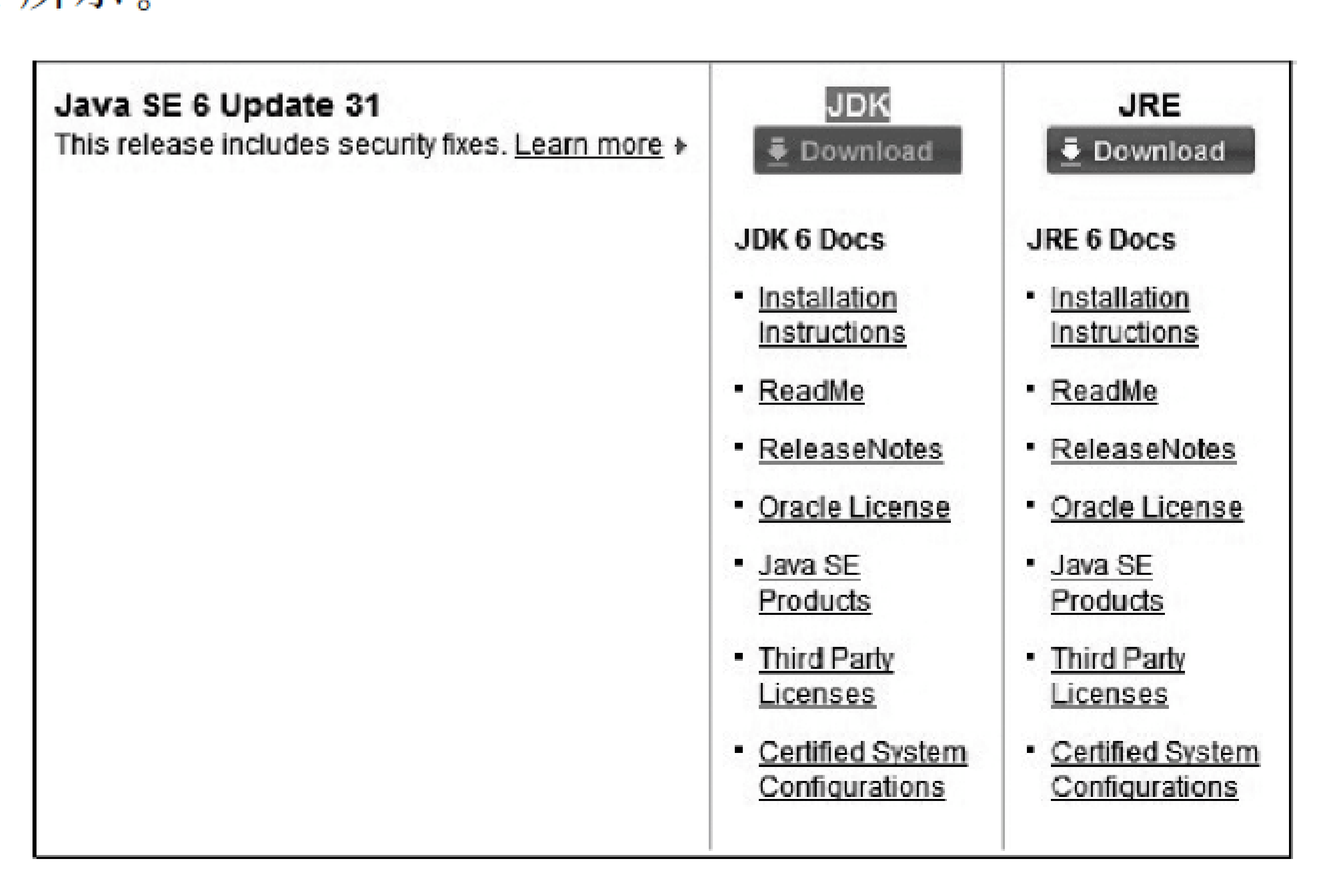


图 5-1 最新版本 JDK 的下载页面

(2) 单击 JDK Download 进入具体的下载页面，即进入 Java SE Downloads 页面，如图 5-2 所示。

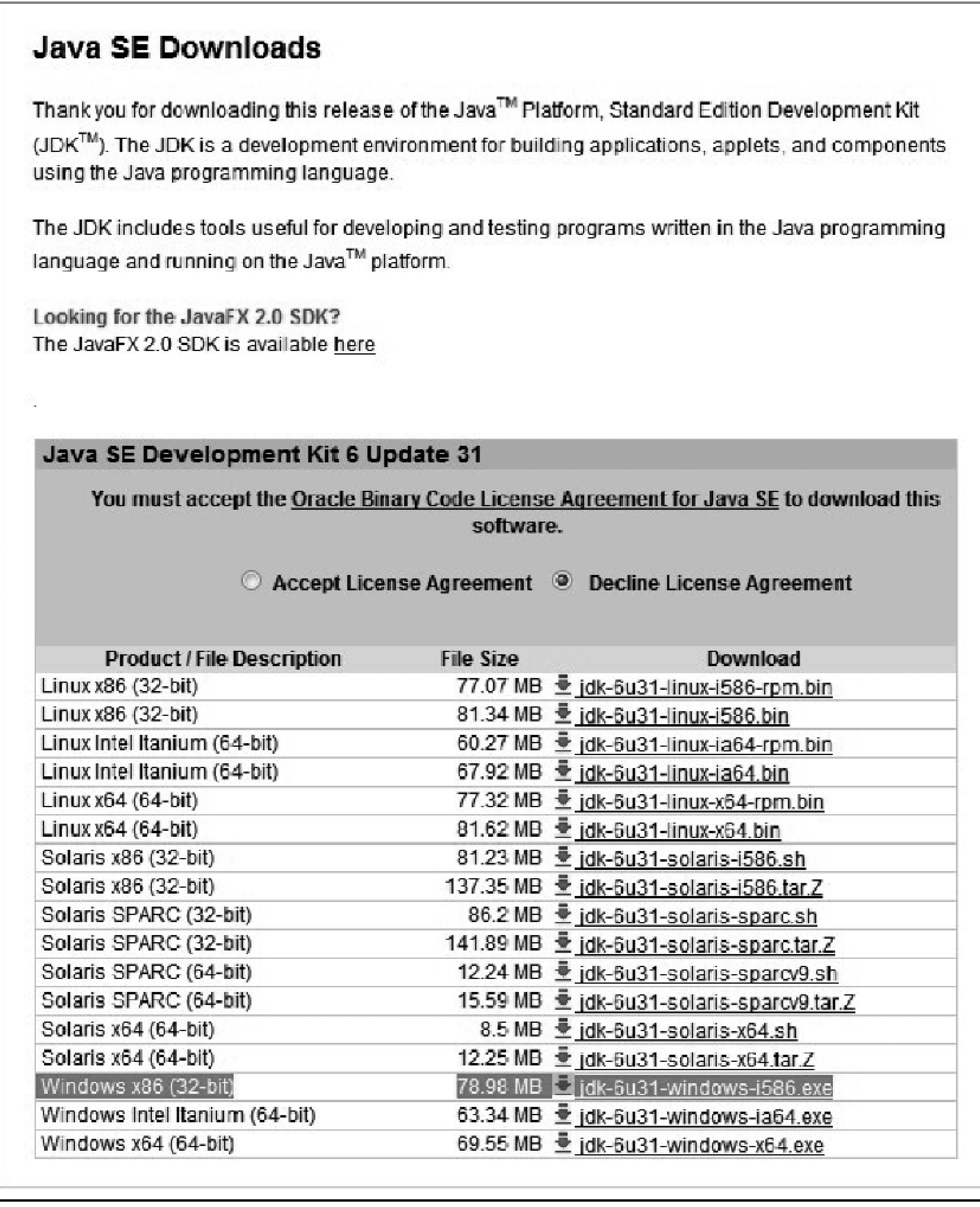


图 5-2　Java SE Downloads 页面

(3) 选择 Windows x86(32-bit)，文件大小约为 78.98MB，名字为 jdk-6u31-windows-i586.exe(个人计算机是 32 位的)。

可以通过多种途径查询自己的计算机到底是 32 位还是 64 位的，这里介绍一种方式仅供参考。打开计算机的系统属性窗口，如图 5-3 所示。

在图 5-3 中可以看到“系统类型：32 位操作系统”，这就可以确定个人计算机是 32 位的。

(4) 接下来可以下载 jdk-6u31-windows-i586.exe 文件了，图 5-4 所示为 360 浏览器下载界面。

2) 安装 JDK

(1) 下载完成后，进入 jdk-6u31-windows-i586.exe 安装包所在目录并打开这个安装文



图 5-3 系统属性窗口

件包，开始安装 Java(TM)SE Development Kit 6 Update 31，Windows 7 先进行配置，如图 5-5 所示。



图 5-4 360 下载 jdk-6u31-windows-i586.exe

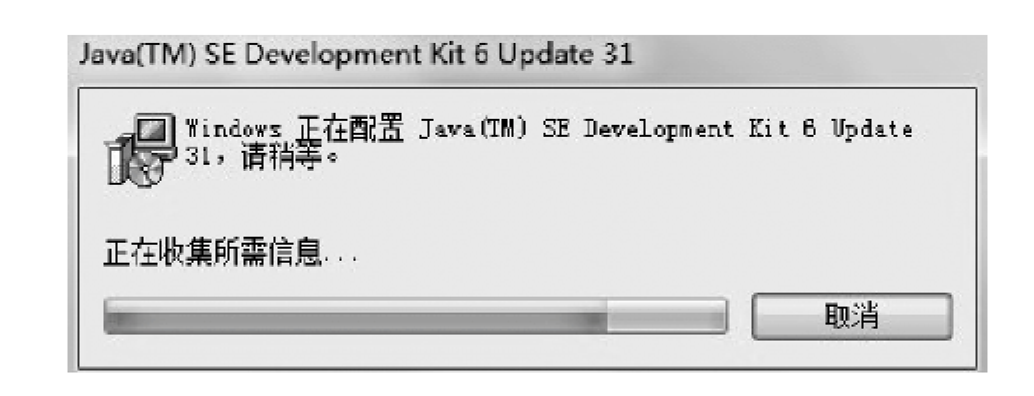


图 5-5 Java(TM)SE Development Kit 6 Update 31 配置

(2) 进入 Java(TM)SE Development Kit 6 Update 31 安装页面，如图 5-6 所示。

(3) 选择安装内容和安装路径，如图 5-7 所示。

(4) 安装过程如图 5-8 所示。



图 5-6　Java(TM)SE Development Kit 6 Update 31 欢迎界面



图 5-7　Java 安装内容和安装路径



图 5-8　安装过程

(5) 安装完成界面如图 5-9 所示。



图 5-9 安装完成界面

在整个安装过程，为了方便使用，一直按照默认路径安装 JDK，将其安装在系统盘 C 盘中，因此最终得到的默认路径为 C:\Program Files\Java\jdk1.6.0_31。

3) 设置环境变量

Java 安装成功后，开始设置环境变量，具体过程如下依次执行“我的电脑”→“属性”→“高级系统设置”→“系统”→“环境变量”→“Hua 的用户变量”，添加以下环境变量。

(1) JAVA_HOME=C:\Program Files\Java\jdk1.6.0_31(指向 JDK 的目录，在这个路径下应该能够找到 bin、lib 等目录)，如图 5-10 所示。

(2) CLASSPATH=.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar(CLASSPATH 为 java 加载类路径，只有在 CLASSPATH 中 java 命令才能识别)，如图 5-11 所示。





图 5-10 JAVA_HOME 变量



图 5-11 CLASSPATH 变量

(3) PATH = %JAVA_HOME%;%PATH%(指向 JDK 的 bin 目录，这样在控制台下面编译、执行程序就不需要再输入一大串路径了)，如图 5-12 所示。

注意：在安装 Java 的过程中，前面三步设置环境变量对搭建整个 Android 开发环境不

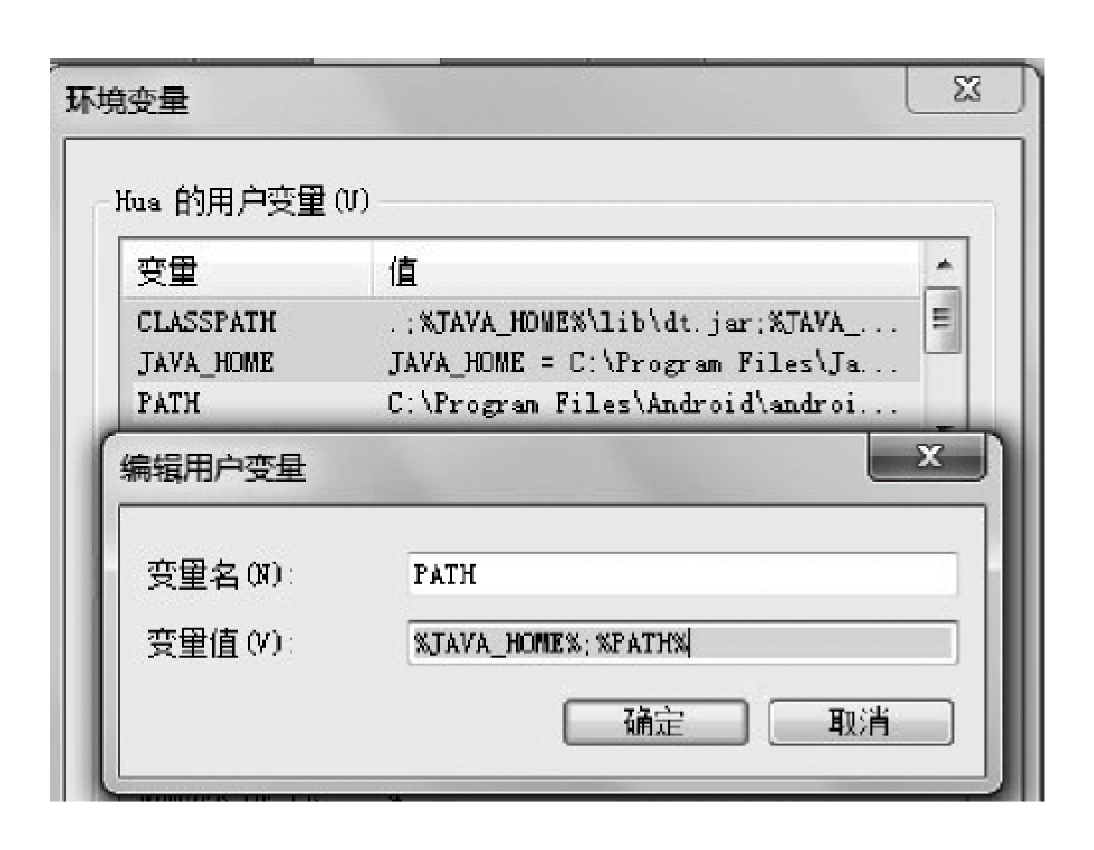


图 5-12　PATH 变量

是必须的,有时候可以直接跳过。

4) 检查 JDK 是否安装成功

安装完成之后,可以检查 JDK 是否安装成功。打开 cmd 窗口,输入“java-version”查看 JDK 的版本信息,即输入命令行:

```
C:\Users\Hua> java - version
```

如果 DOS 系统出现如图 5-13 所示的界面,这就表明 JDK 已经安装成功了,因为可以清楚读出 Java 的详细版本信息。

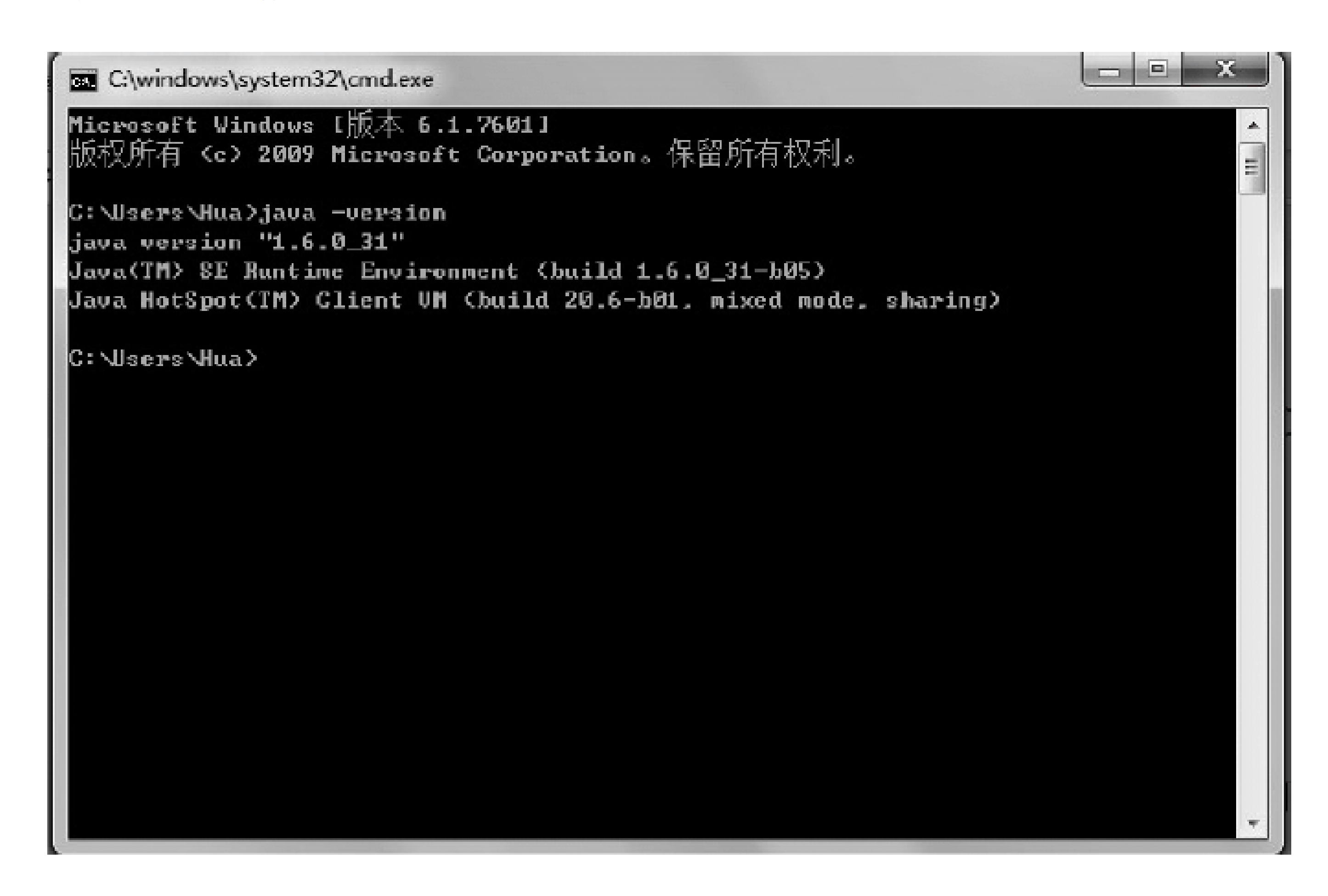


图 5-13　验证 JDK 安装是否成功

2. Eclipse 安装

关于 Eclipse 的安装前文已有介绍,这里简要介绍另一款 Eclipse 的下载、安装过程。在 IE 浏览器中输入网页地址 http://www.eclipse.org/downloads/,进入 Eclipse 下载页面,如图 5-14 所示。

这里选择第一项,即 Eclipse IDE for Java EE Developers,然后选择其右侧的 Windows 32 Bit 选项,单击下载按钮开始下载 Eclipse。下载详细页面图如图 5-15 所示。



图 5-14 Eclipse Downloads 页面



图 5-15 Eclipse 下载页面

下载完成后，将它进行解压到相关的路径。这个路径自己可以自由选择，但一般建议不放在系统盘 C 盘中，因为这个压缩文件比较大，下载需要的时间比较长，C 盘不备份，遇上计算机故障容易造成文件丢失；可以放在 D 盘(在本次搭建开发环境时就将其放在 D 盘中)。这个 Eclipse 不需要安装，解压后可以直接使用，这就是它自身最大的便利之处，而且不受路径的影响。解压后的 eclipse 文件夹如图 5-16 所示，其中黑色框加注的 eclipse. exe 就是所要得到的文件，可以直接单击运行它。

3. Android SDK 安装

Android SDK 有两种下载版本，一种是包含了具体版本的 SDK，另一种只是一个升级工具而不包含具体的 SDK 版本，前者大概有 70MB，而后者一般只有 20MB 左右，因此一般建议下载后面一种升级包。

1) 下载 Android SDK

直接在浏览器中输入网页地址 http://developer. android. com/sdk/index. html，进入 Android Developers 页面，如图 5-17 所示。



图 5-16　eclipse 文件夹

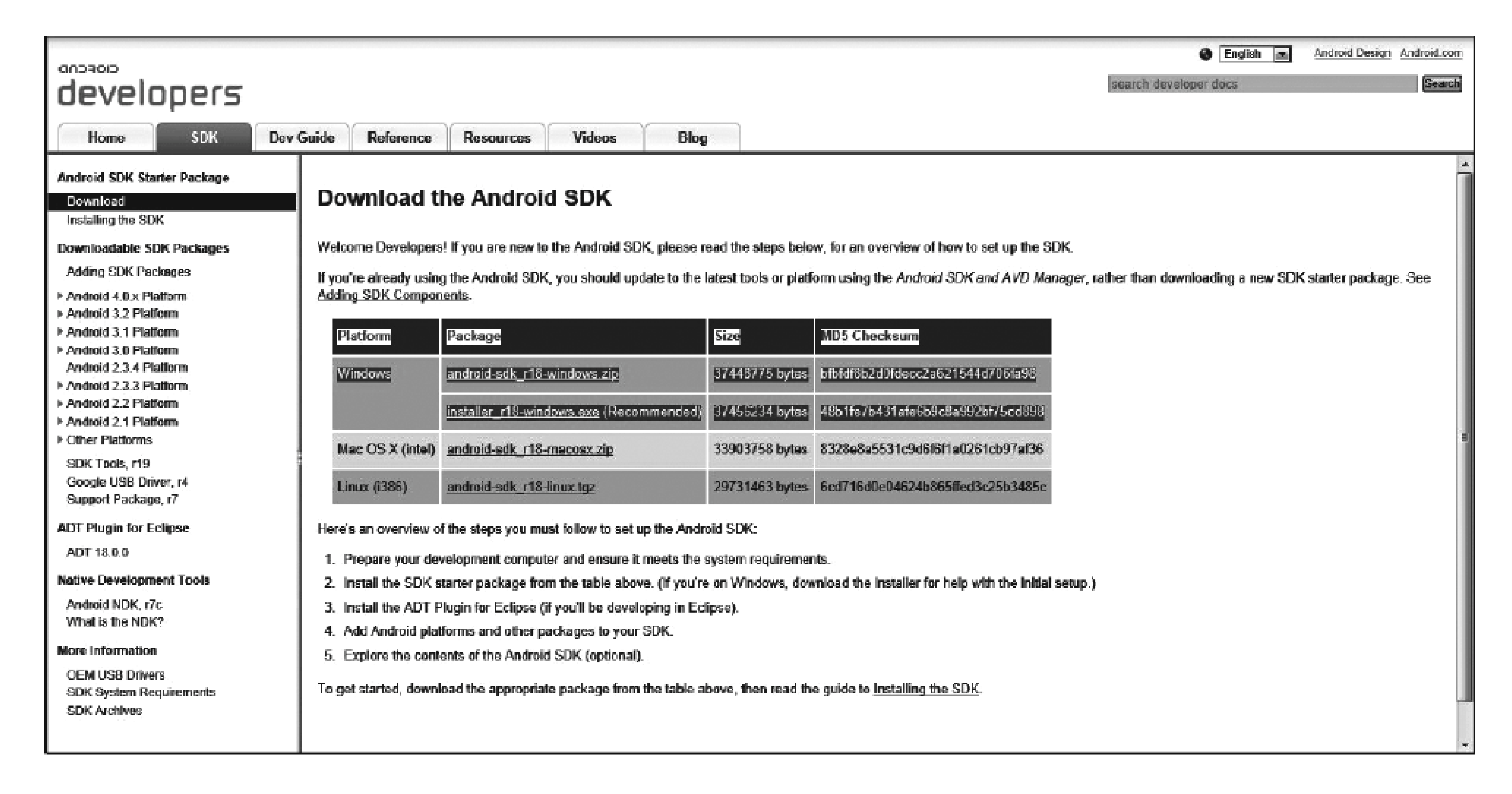


图 5-17　Android 下载界面

在标题为 Download the Android SDK 页面，选择 Platform 为 Windows、Package 为 installer r18-windows. exe 开始下载；下载完成后将其解压到任意路径，一般建议放在计算机的 D 盘中。另外，MAC OS X (intel)主要是安装在 Apple 公司生产的 MAC 机上的；学习 Linux 的读者同样也可以下载一个 Linux 版的 Android SDK 进行开发使用，在 Linux 系统下构建的 Android 系统应用范围很广泛。

2）安装 Android SDK

运行 installer r18-windows. exe 开始进行 Android SDK 安装。

(1) Android SDK Tools Setup 欢迎界面，如图 5-18 所示。



图 5-18　Android SDK Tools Setup 欢迎界面

(2) 单击 Next 按钮，寻找 JDK 安装目录，如图 5-19 所示。

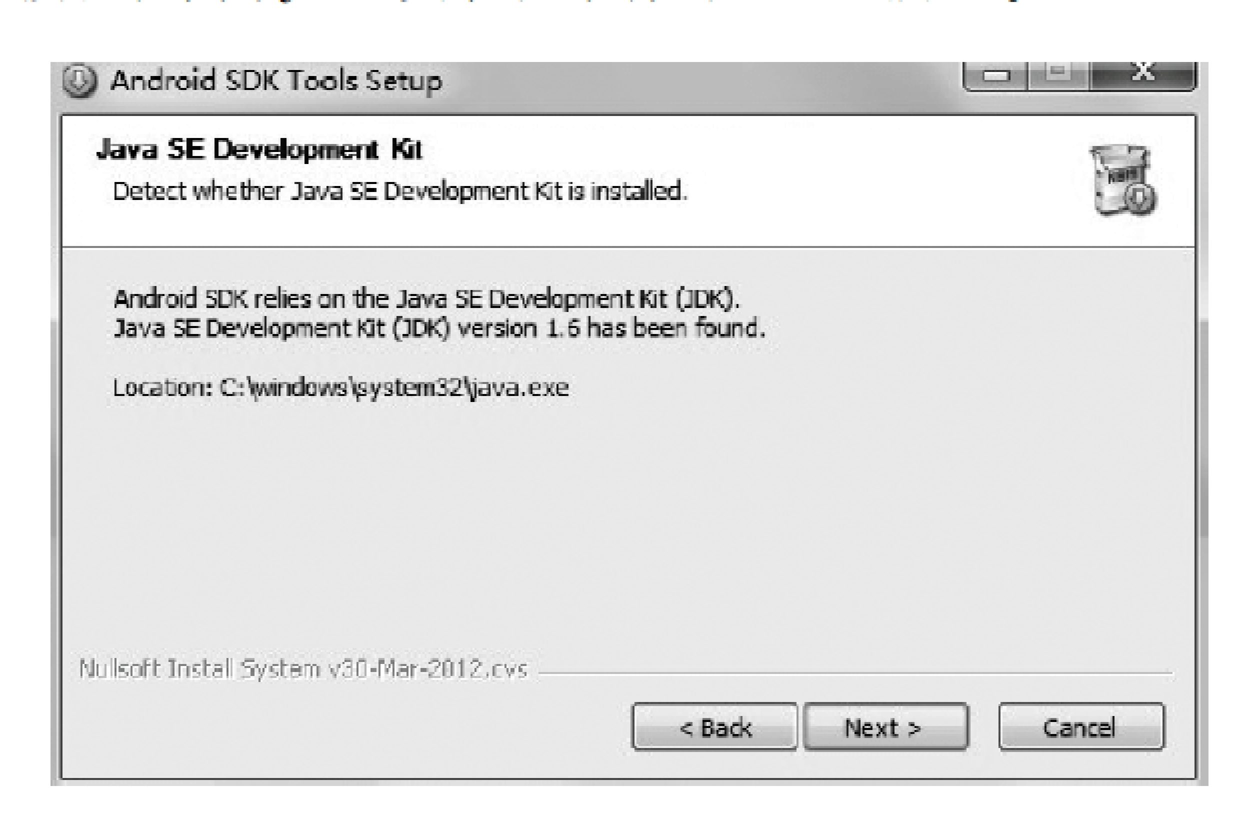


图 5-19　寻找 JDK 安装目录

(3) 单击 Next 按钮，选择 Android SDK 安装位置，如图 5-20 所示。

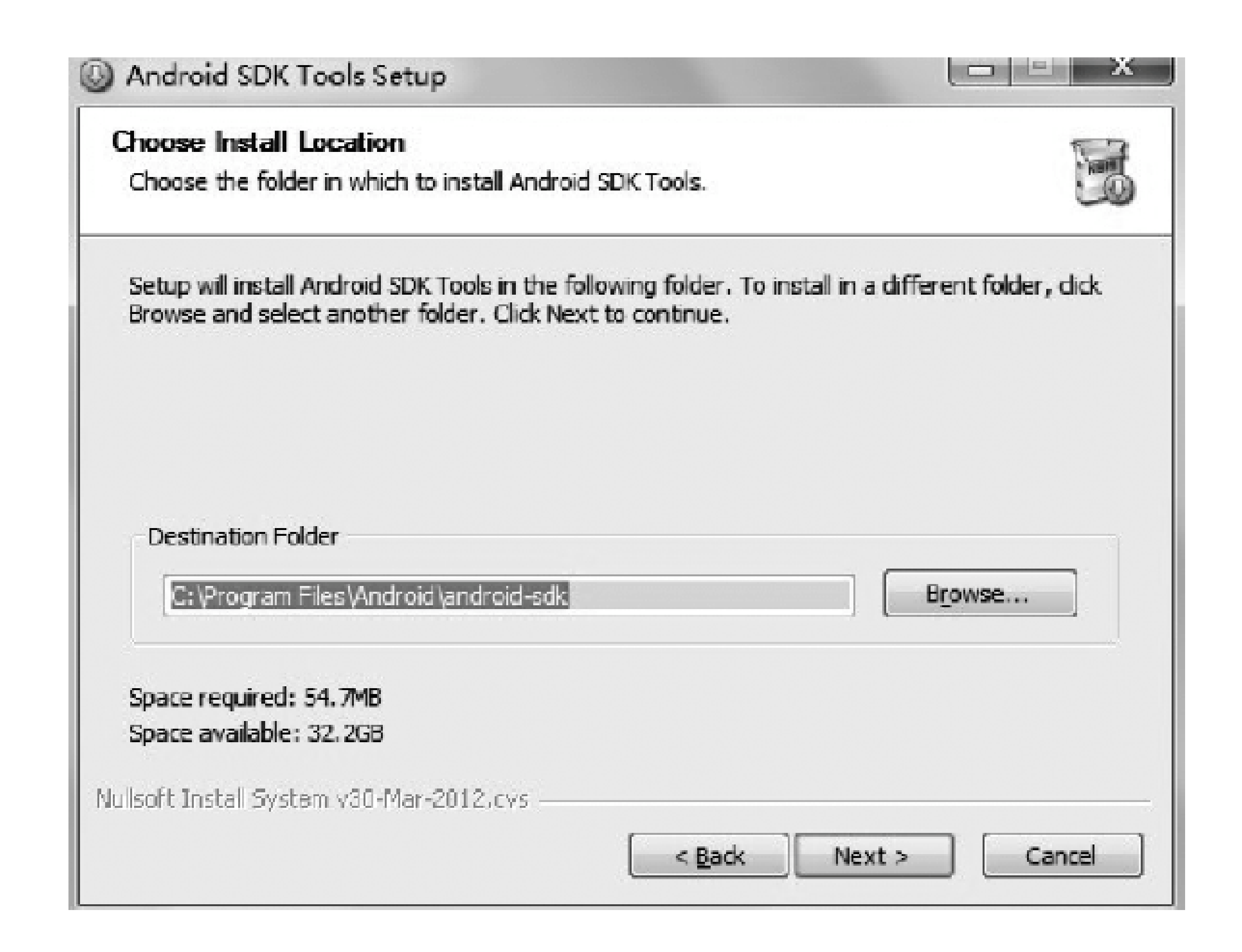


图 5-20　选择 Android SDK 安装位置

(4) 单击 Next 按钮,选择开始菜单夹,如图 5-21 所示。

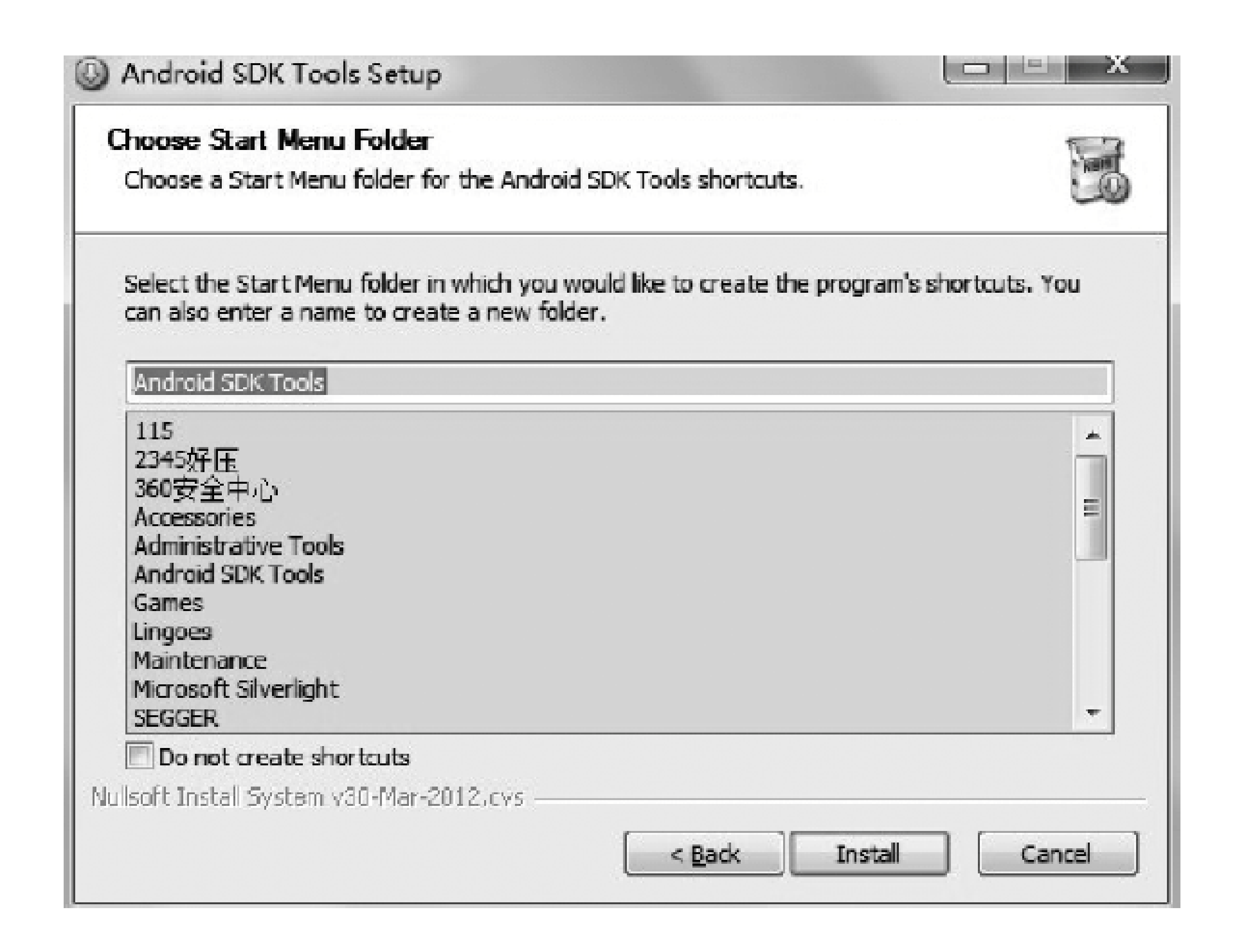


图 5-21　选择开始菜单夹

(5) 单击 Next 按钮,安装解压过程如图 5-22 所示。

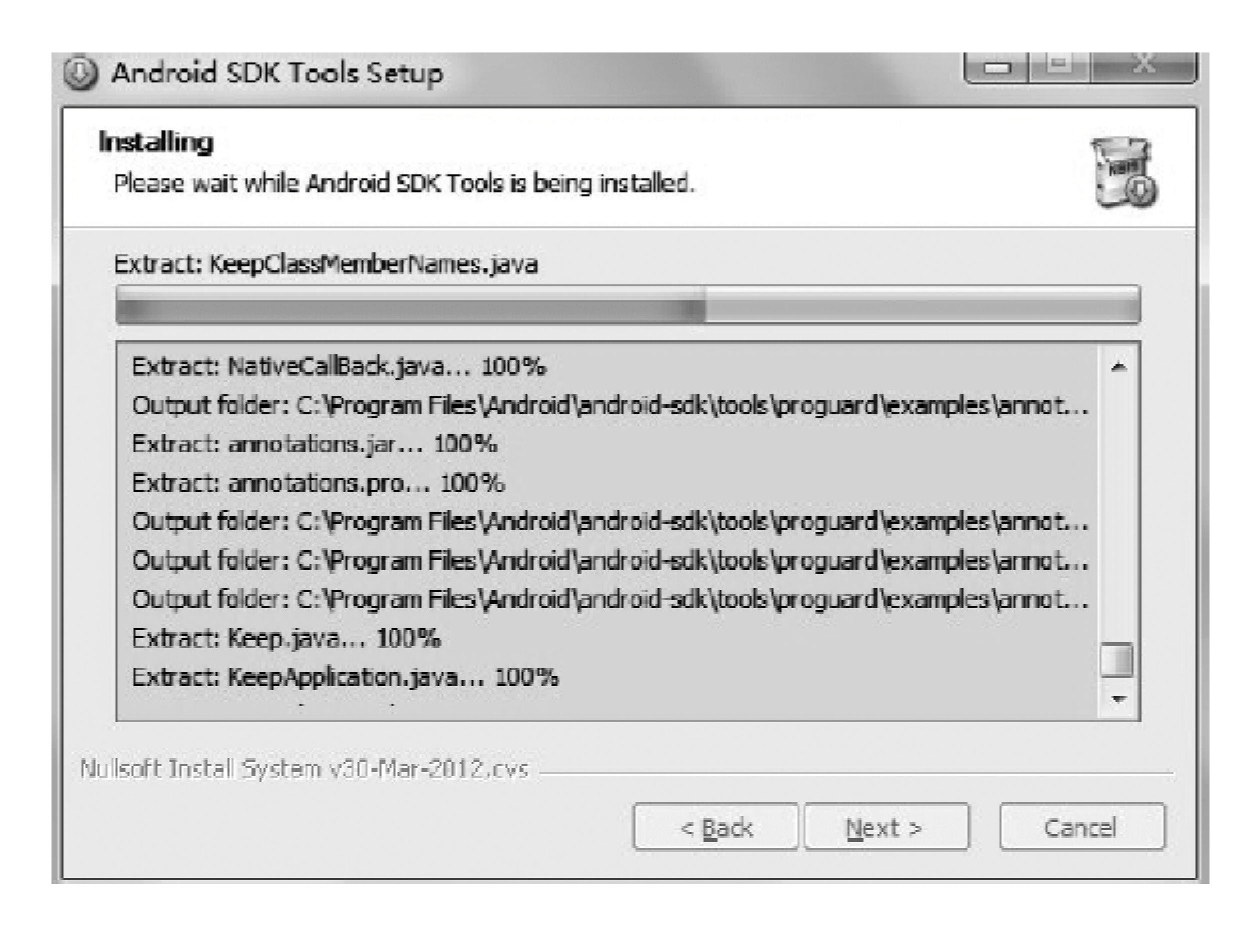


图 5-22　安装解压过程

(6) Android SDK 安装完成界面如图 5-23 所示。



图 5-23　Android SDK 安装界面

3）Android SDK Manager

（1）整个 Android SDK 安装完成后，单击图 5-23 中的 Finish 按钮会自动启动 Android SDK Manager 界面，如图 5-24 所示。

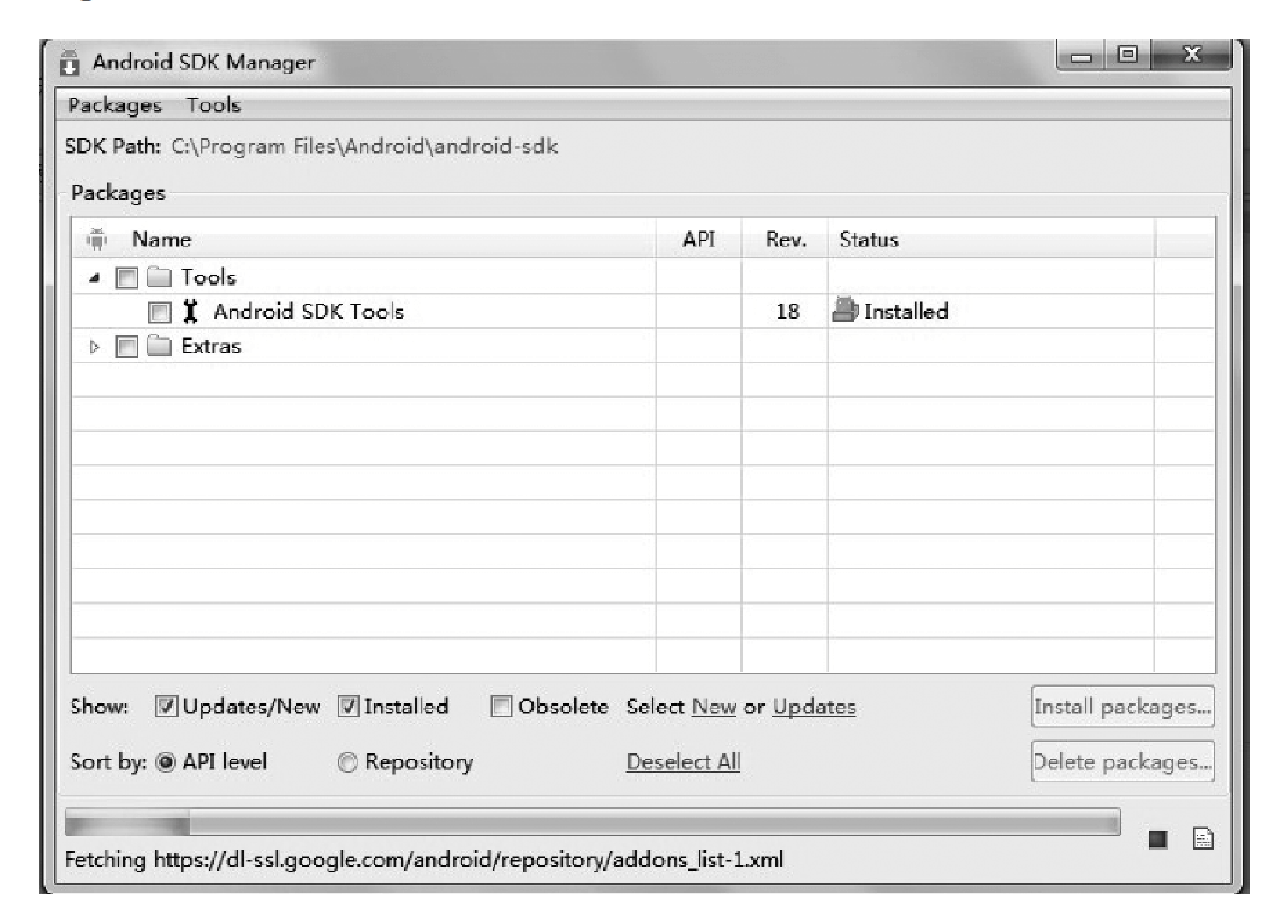


图 5-24　Android SDK Manager 界面

（2）获取 Android 版本信息后的 Android SDK Manager 页面如图 5-25 所示。

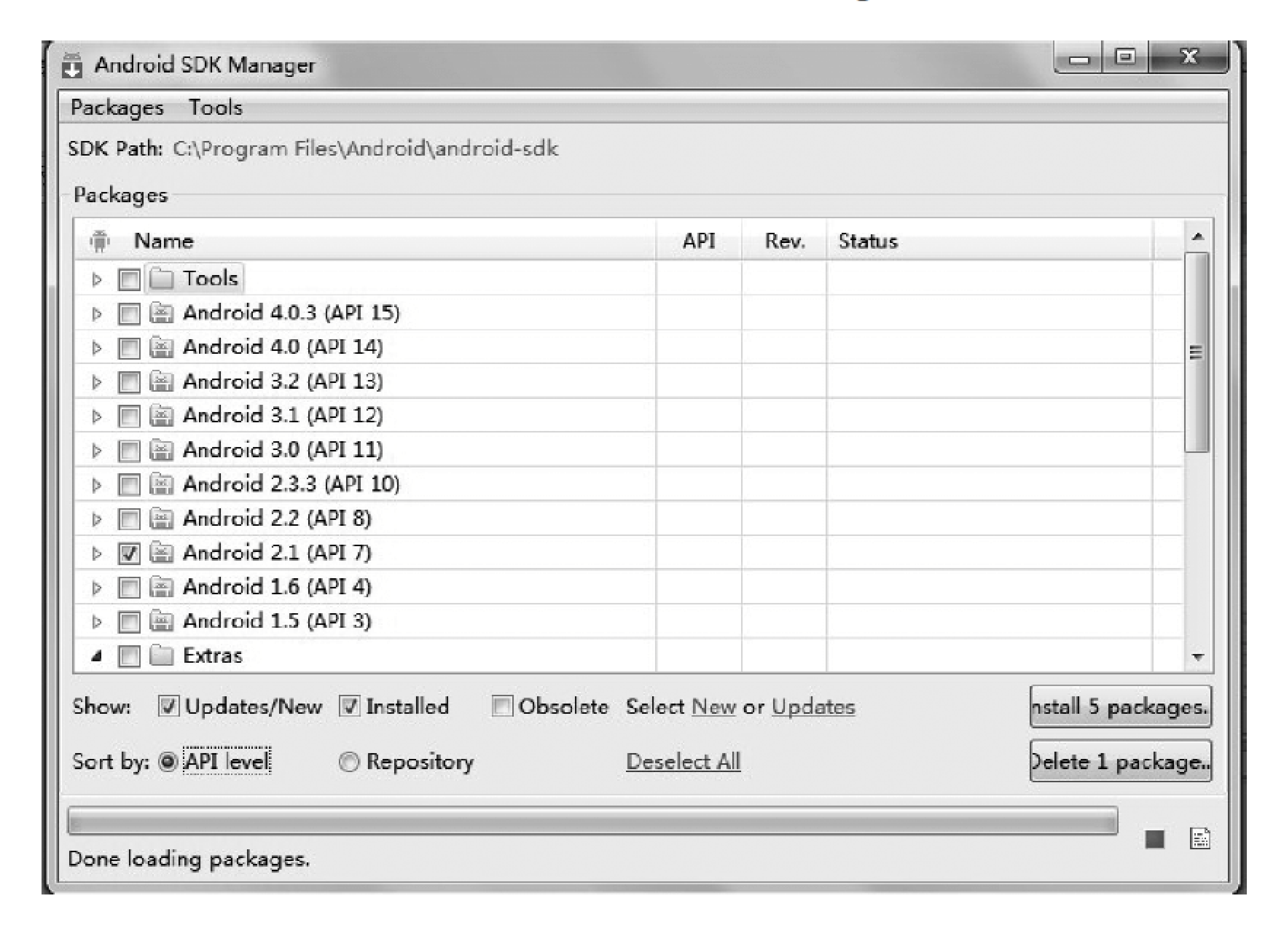


图 5-25　获取 Android 版本信息

4）安装 Android 2.1

在图 5-25 中选择 Android 2.1(API 7)进行下载安装，主要下载的内容为：

(1) Android SDK Tools，revision 19。

(2) Android SDK Platform-tools，revision 1。

(3) SDK Platform Android 2.1 API 7，revision 1。

(4) Samples for SDK API，revision 1。

也可以通过右侧界面的 Accept 或 Reject 选择要安装的内容，如图 5-26 中左边对话框中的 Google APIs，Android API 7，revision 1 和 Google USB Driver，revision 4，如果全部接受可以直接选择右侧的 Accept All 单选按钮，如图 5-26 所示。

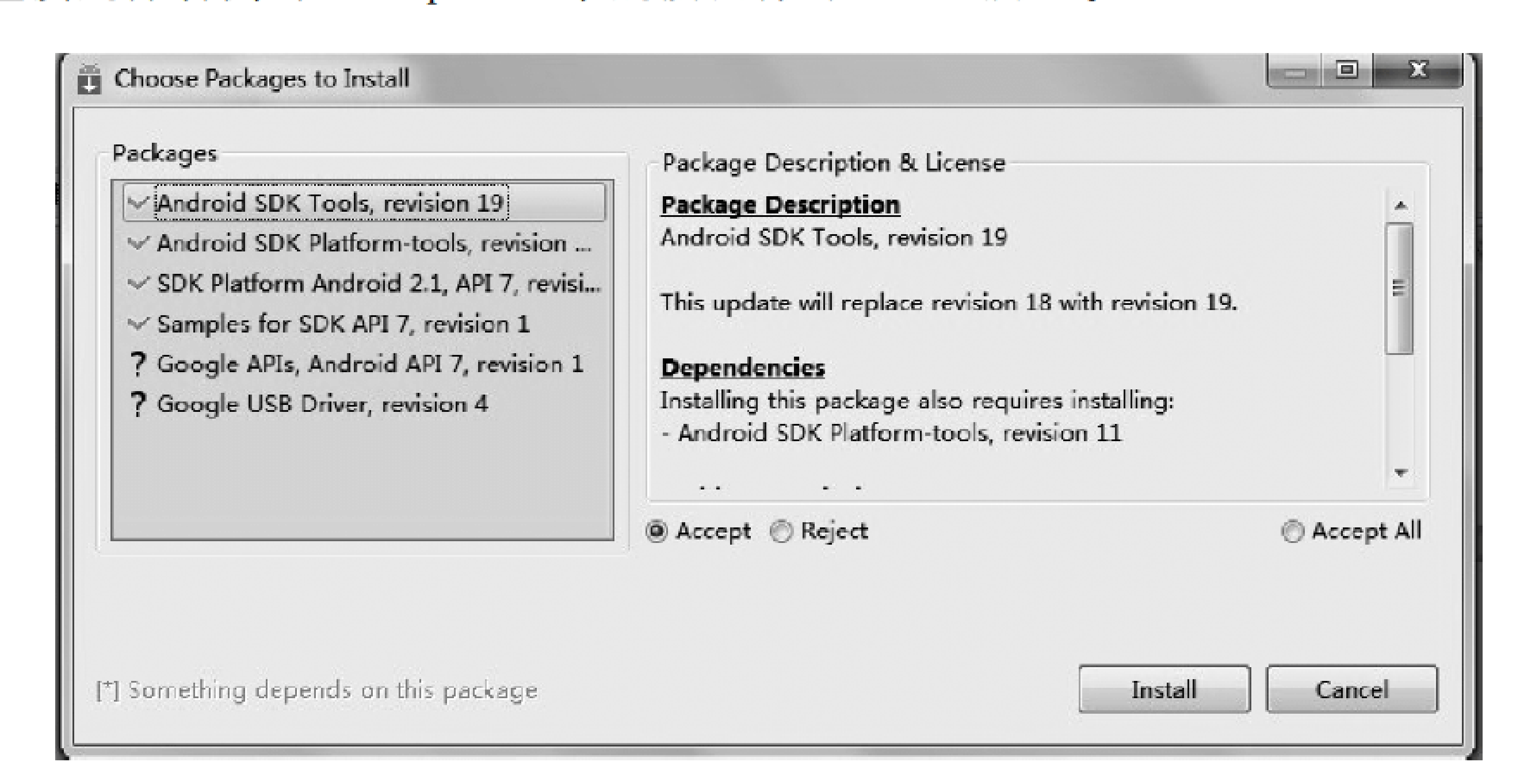


图 5-26　Android 2.1(API 7)下载页面

Android 2.1 安装完成图如图 5-27 所示。这时如果需要的话可以继续下载其他版本的 Android 系统，如果不需要则直接单击对话框右上角的关闭按钮。

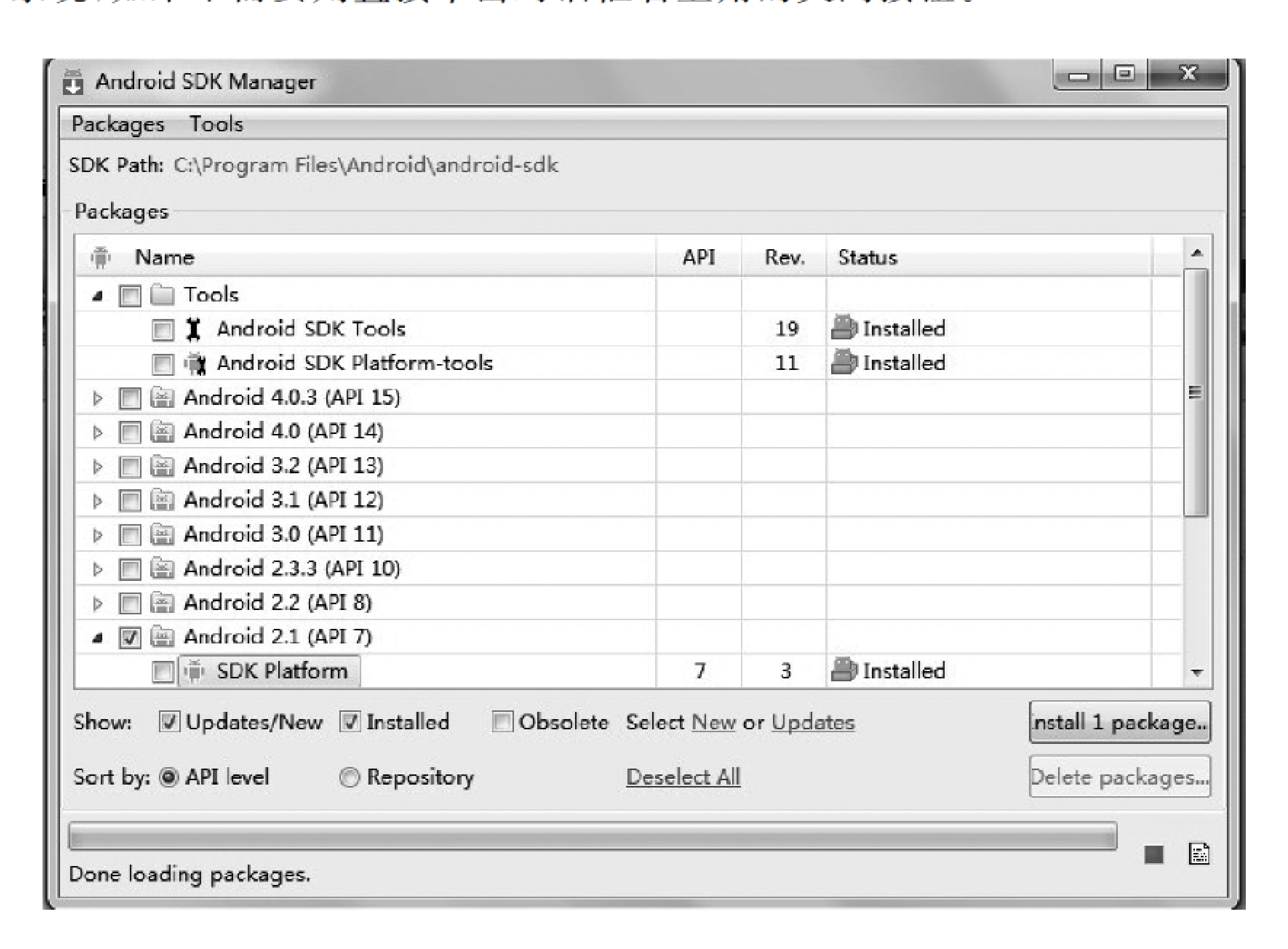


图 5-27　Android 2.1 安装完成

5）设置环境变量

依次执行“我的电脑”→“属性”→“高级系统设置”→“高级”→“环境变量”，添加以下环境变量：

用户变量中新建 PATH 值为 Android SDK 中的 tools 绝对路径，这里为 C:\Program Files\Android\android-sdk。

这个用户变量不同于前面 Java 环境的变量设置，因为 Java 环境的变量设置在整个 Android 搭建中可有可无，它只是在 Java 版本检测过程或者说单纯 Java 开发环境中需要添加。但这里的环境变量的设置却是不可或缺的，因为若缺少了这个 PATH 则 Android Emulator 将无法正常工作。具体的变量设置如图 5-28 所示。



图 5-28　PATH 设置

6）检查 Android SDK 是否安装成功

两次单击“确定”按钮后，需要重新启动计算机。计算机重启后，进入 cmd 命令窗口，输入命令行：

```
C:\Users\Hua>cd c:\program files\android\android-sdk\tools
C:\Program Files\Android\android-sdk\tools>android -h
```

运行 android -h，如果有类似图 5-29 所示的输出，则表明 Android SDK 安装成功。

4. ADT 安装

(1) 打开 Eclipse IDE，执行菜单 Help→Install New Software 命令，打开 Install 对话框。

(2) 单击 Add 按钮弹出 Add Site 对话框，要求输入 Name 和 Location。Name 自己随便取，这里取名为 Android 以便辨认；Location 文本框中输入 https://dl-ssl.google.com/android/eclipse，如图 5-30 所示。

```
D:\Android\android-sdk\tools>android.bat -h

       Usage:
       android [global options] action [action options]
       Global options:
  -h --help       : Help on a specific command.
  -v --verbose    : Verbose mode, shows errors, warnings and all messages.
     --clear-cache: Clear the SDK Manager repository manifest cache.
  -s --silent     : Silent mode, shows errors only.

                                                                       Valid
                                                                       actions
                                                                       are
                                                                       composed
                                                                       of a verb
                                                                       and an
                                                                       optional
                                                                       direct
                                                                       object:
-    sdk              : Displays the SDK Manager window.
-    avd              : Displays the AVD Manager window.
-   list              : Lists existing targets or virtual devices.
-   list avd          : Lists existing Android Virtual Devices.
-   list target       : Lists existing targets.
-   list sdk          : Lists remote SDK repository.
- create avd          : Creates a new Android Virtual Device.
-   move avd          : Moves or renames an Android Virtual Device.
- delete avd          : Deletes an Android Virtual Device.
- update avd          : Updates an Android Virtual Device to match the folders
                        of a new SDK.
- create project      : Creates a new Android project.
- update project      : Updates an Android project (must already have an
                        AndroidManifest.xml).
- create test-project : Creates a new Android project for a test package.
- update test-project : Updates the Android project for a test package (must
                        already have an AndroidManifest.xml).
- create lib-project  : Creates a new Android library project.
- update lib-project  : Updates an Android library project (must already have
                        an AndroidManifest.xml).
- create uitest-project: Creates a new UI test project.
- update adb          : Updates adb to support the USB devices declared in the
                        SDK add-ons.
- update sdk          : Updates the SDK by suggesting new platforms to install
                        if available.
```

图 5-29　验证 Android SDK 是否安装成功

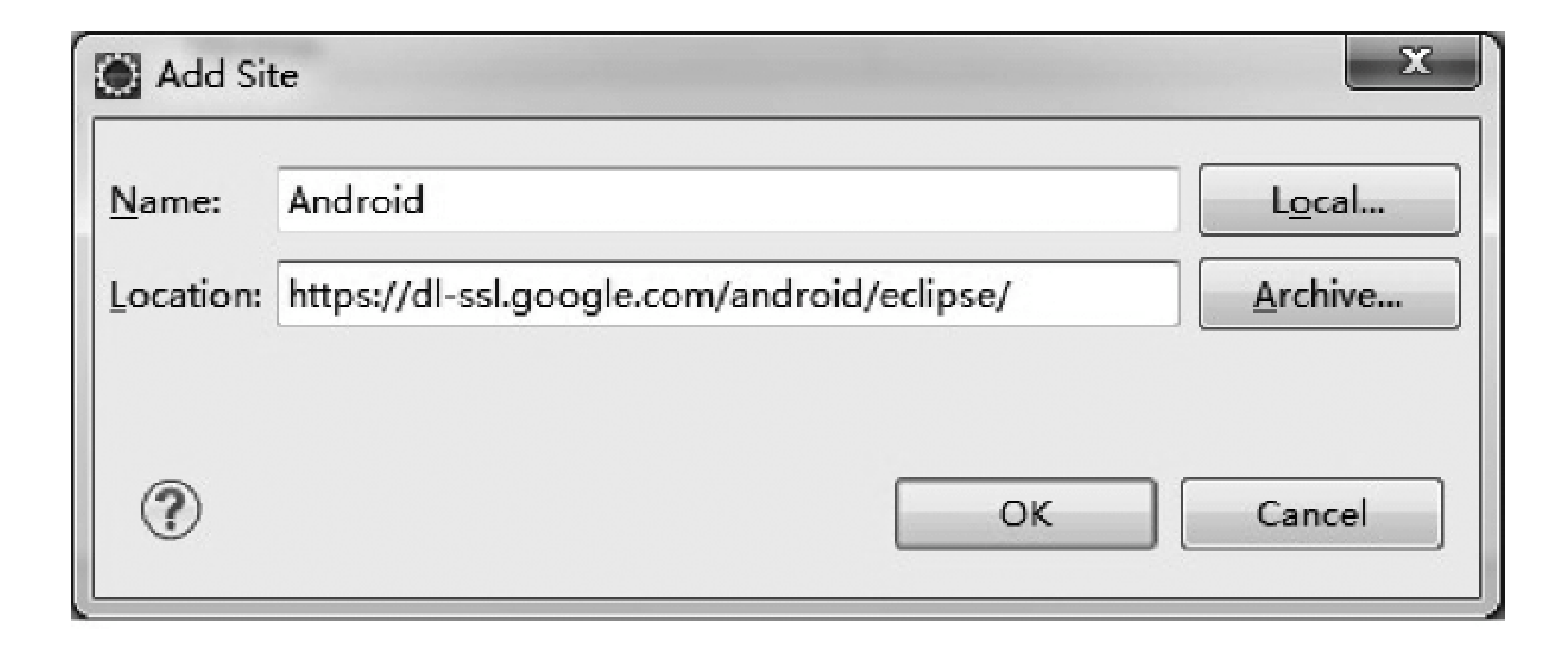


图 5-30　Add Site

(3) 单击 OK 按钮返回后，在 Work with 后的下拉列表框中选择刚才添加的 ADT，这时会看到下面出现 Developer Tools，展开它会有 Android DDMS 和 Android Development Tools 复选框，选中它们，如图 5-31 所示。

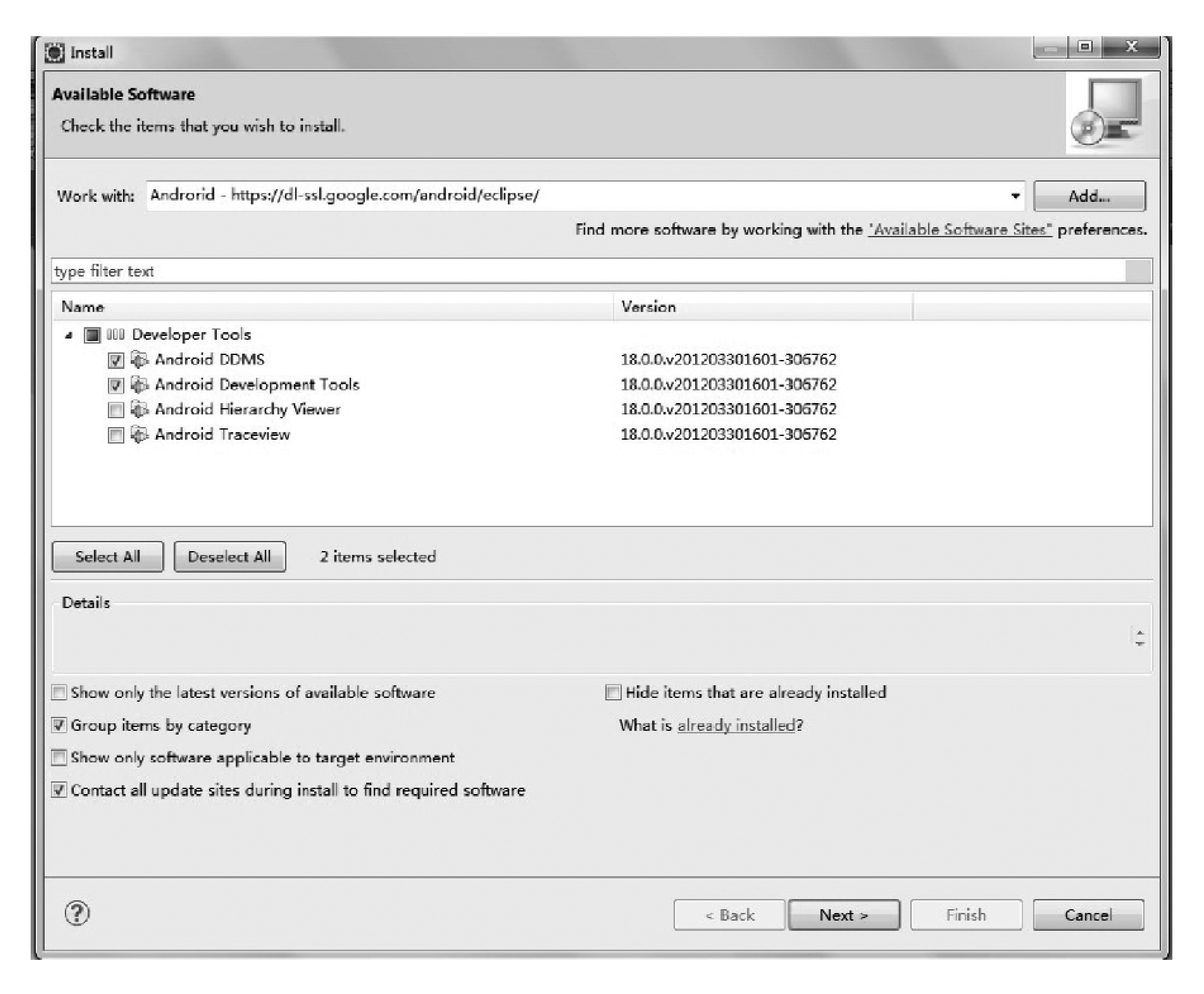


图 5-31　选择 Developer Tools

(4) 安装 Android DDMS 和 Android Development Tools 是一个比较慢的过程。由于需要从 Google 服务器上下载这两个安装包，网络速度不快的情况下大概需要 1 个小时的时间。下载完成后安装过程图如图 5-32 所示。

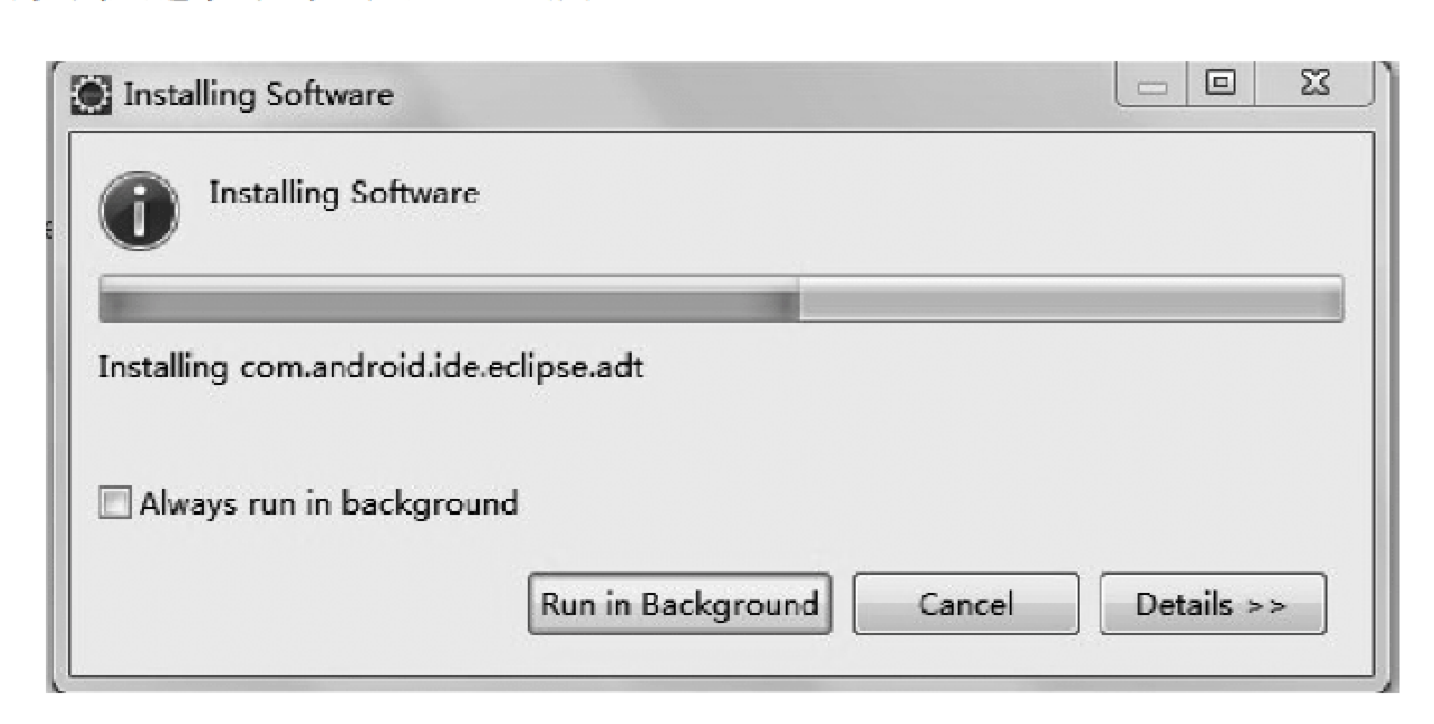


图 5-32　正在安装软件

(5) Android DDMS 和 Android Development Tools 安装完成之后，会出现如图 5-33 所示的界面，要求配置 SDK。

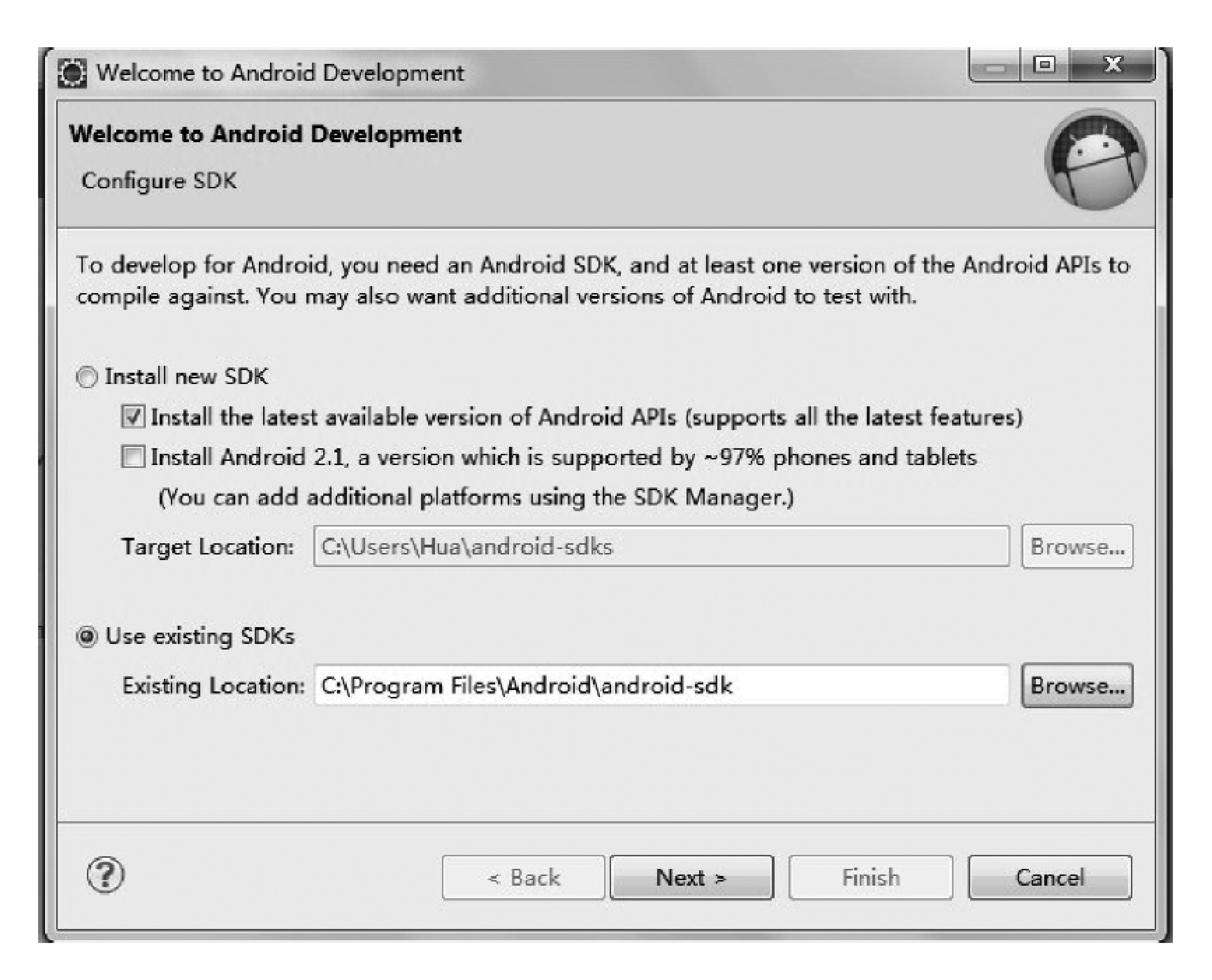


图 5-33　配置 SDK

（6）在图 5-33 中选中 Use existing SDKs 单选按钮，单击 Existing Location 右侧的 Browse 按钮并选中 SDK 路径，本机为：C:\Program Files\Android\android-sdk。

（7）单击 Next 按钮，出现 Contribute Usage Statistics 界面，如图 5-34 所示。

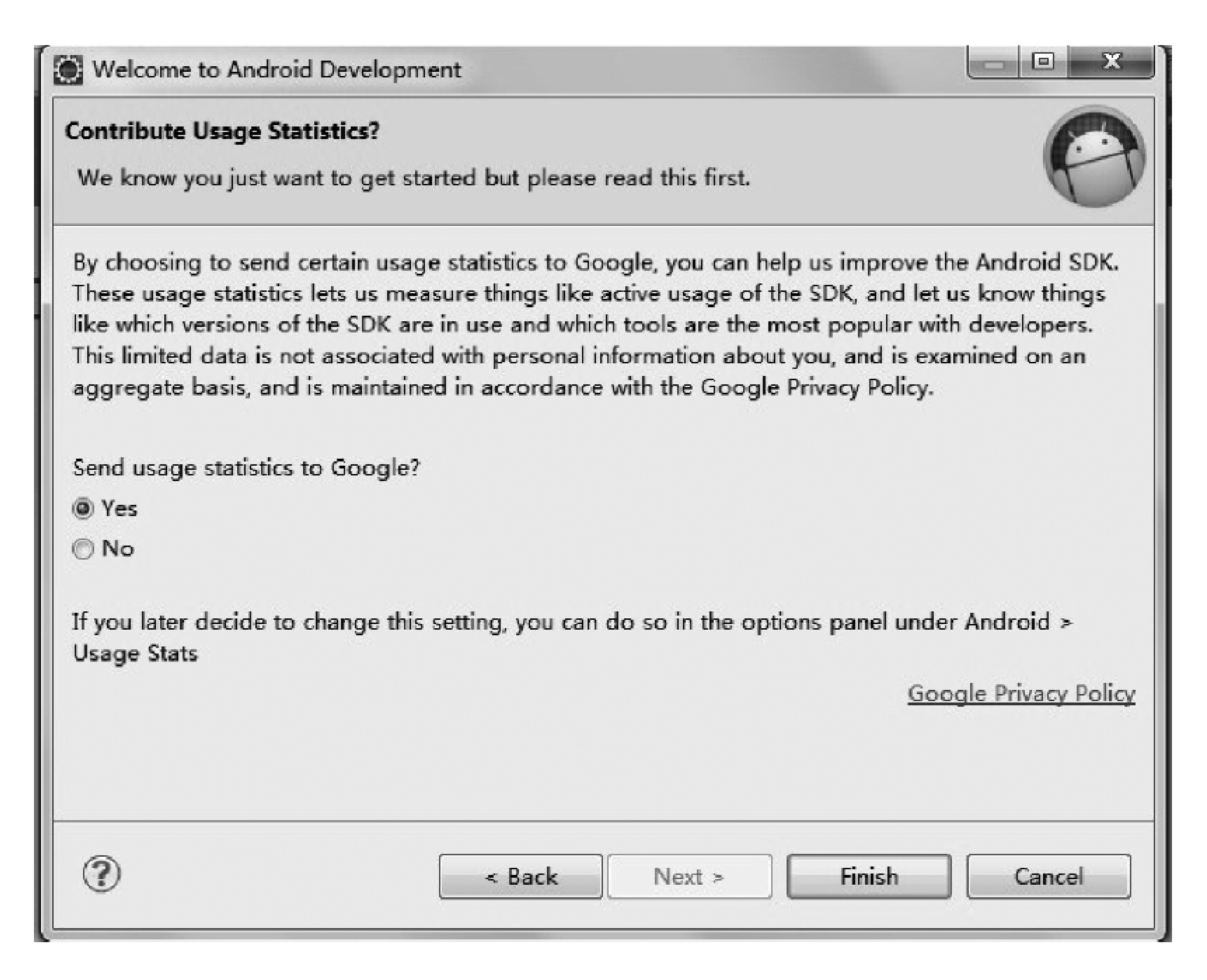


图 5-34　Contribute Usage Statistics 界面

(8) 选中 Yes 单选按钮，最后单击 Finish 按钮就可以了。

5. 创建 AVD

为使 Android 应用程序可以在 Emulator 上运行，就必须创建 AVD(Android Virtual Device)。创建 AVD 的操作步骤如下：

(1) 打开 Eclipse，选择 Window→AVD Manager 菜单命令，打开 Android Virtual Device Manager 对话框，如图 5-35 所示。

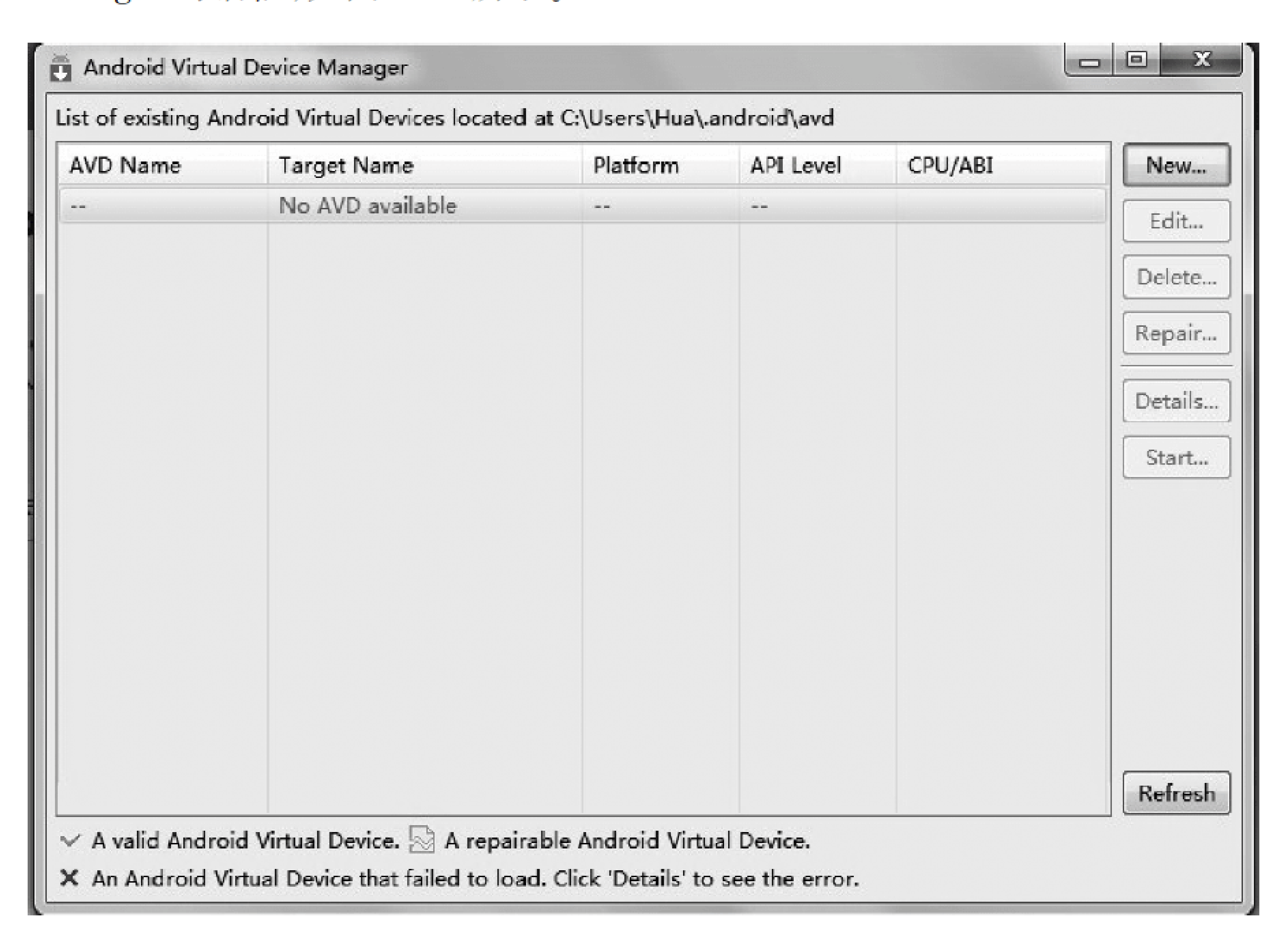


图 5-35　Android Virtual Device Manager 对话框

(2) 单击右侧面板上的 New 按钮，打开 Create new Android Virtual Device 对话框，按如图 5-36 所示进行设置。

输入 Name 为 AVD2.1 主要是为了便于记忆；选择 Target 为 Android 2.1 - API Level 7，因为在前面的安装过程中只安装了这个 Android 2.1 版本的安装包，读者可以根据自己所下载的安装包自行决定选择 Android 版本；SD Card 可以选择任意大小值，Skin 任意选，Hardware 目前保持默认值，因为这些变量都不会影响操作过程中的数据，它只是改变了 Android Virtual Devices 界面。

(3) 最后单击 Create AVD 按钮，即可完成创建 AVD 的任务。

当单击图 5-35 中右侧的 New 按钮，出现的 Create Android Virtual Devices 对话框中的 Target 下拉列表框中没有可选项时，需要回到 Eclipse 界面，执行 Window→Android SDK Managers 菜单命令，选择 tools 选项卡，重新开始安装 tools 和 Android 2.1 版本。打开这个界面时出现的网址为 https://dl-ssl.google.com/android/repository/repository.xml，然后选择需要安装的 Android 版本，继续安装，直到完成为止。这样做的原因在于前面安装 Android SDK 时没有安装一些必要的可用包(Available Packages)，导致系统无法识别 Android 2.1 版，这时就是重复上面未完成的工作，把 Android 整个配置的过程完善化、



图 5-36　Create Android Virtual Devices 对话框

具体化。

6. HelloWorld

1）创建 Android 项目 HelloWorld

回到 Eclipse 界面，开始新建一个项目，创建过程与第 2 章中创建 HelloAndroid 项目基本相同，这里只概要介绍。

（1）打开 New Project 对话框，选择 Android 目录下的 Android Project 项，开始建立新项目，如图 5-37 所示。

（2）单击 Next 按钮，进入 New Android Project 对话框，按如图 5-38 所示进行设置。

在这里，Project Name 设置为 HelloWorld；选择 Create new project in workspace 单选按钮；选中 Use default location 复选框，即使用默认路径；因为现在是第一次打开 Eclipse，会弹出设置 workspace 的对话框，默认即可，即路径为 C：/Users/Hua/workspace，这样新建的 Android Project 就会出现在这个目录下面，当然也可以选择一个不一样的 workspace。

在 Use default location 区域允许选择一个已存在的项目。

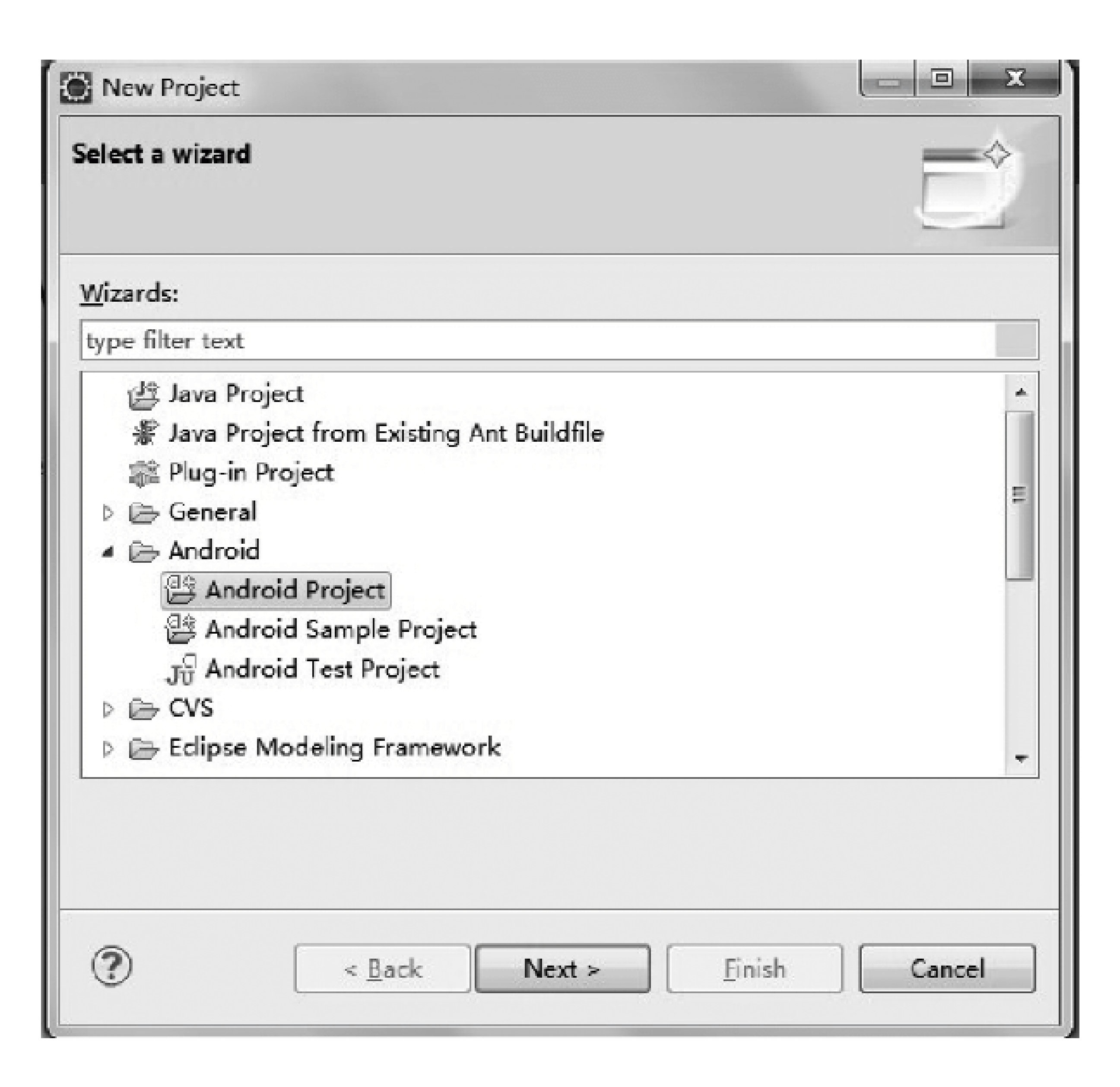


图 5-37　New Project 对话框



图 5-38　New Android Project 对话框

(3) 单击 Next 按钮后，出现 Choose an SDK to target 界面，如图 5-39 所示。

在这里 Target Name 选择 Android 2.1。

(4) 单击 Next 按钮，出现 Configure the new Android Project 界面，如图 5-40 所示。

这里设置 Package Name 为 hello.namespace。可以按照个人意愿随意设置名字，但要注意的是名字首字母要以小写字母开头，不能使用大写字母。其他值保持默认状态就可以了。

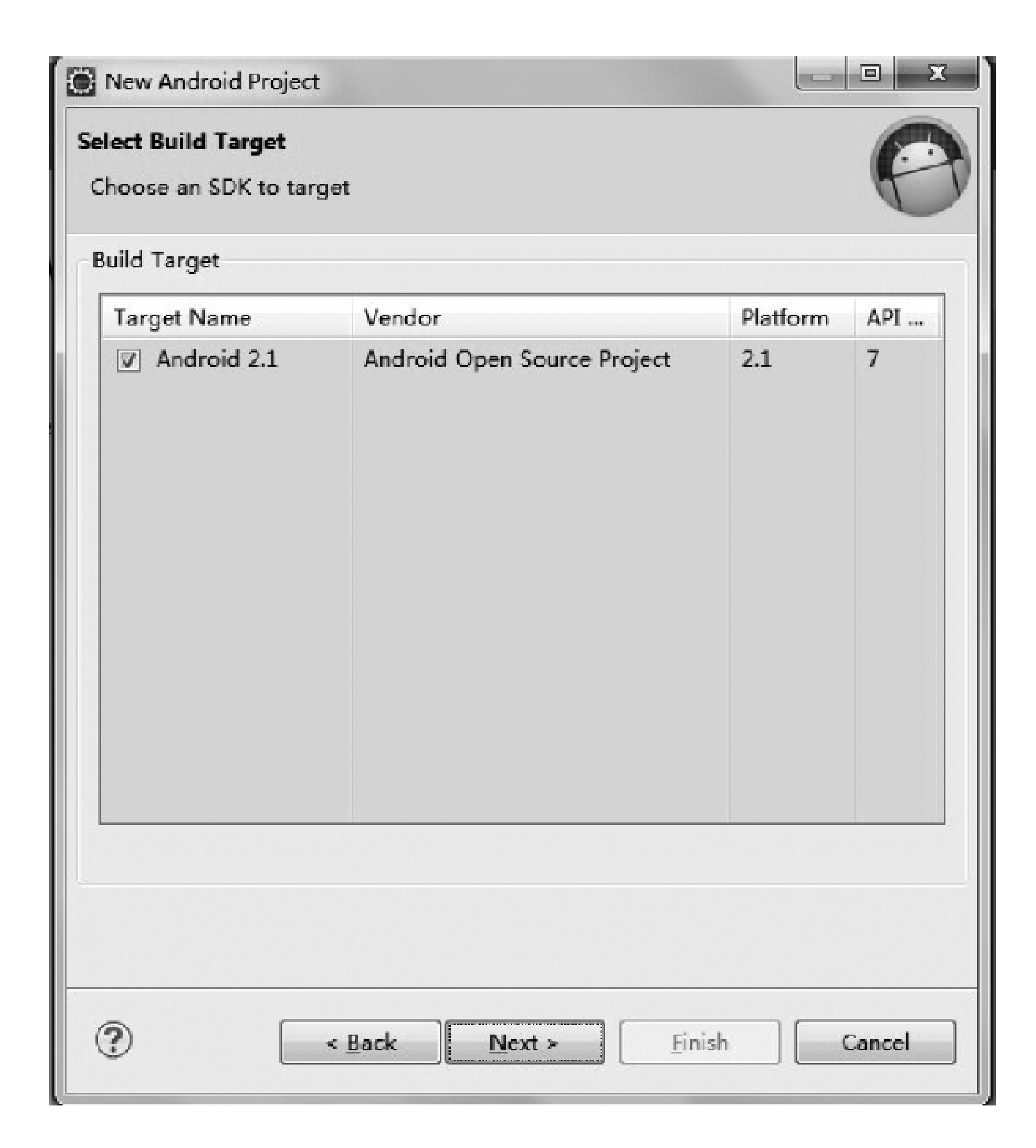


图 5-39　Choose an SDK to target

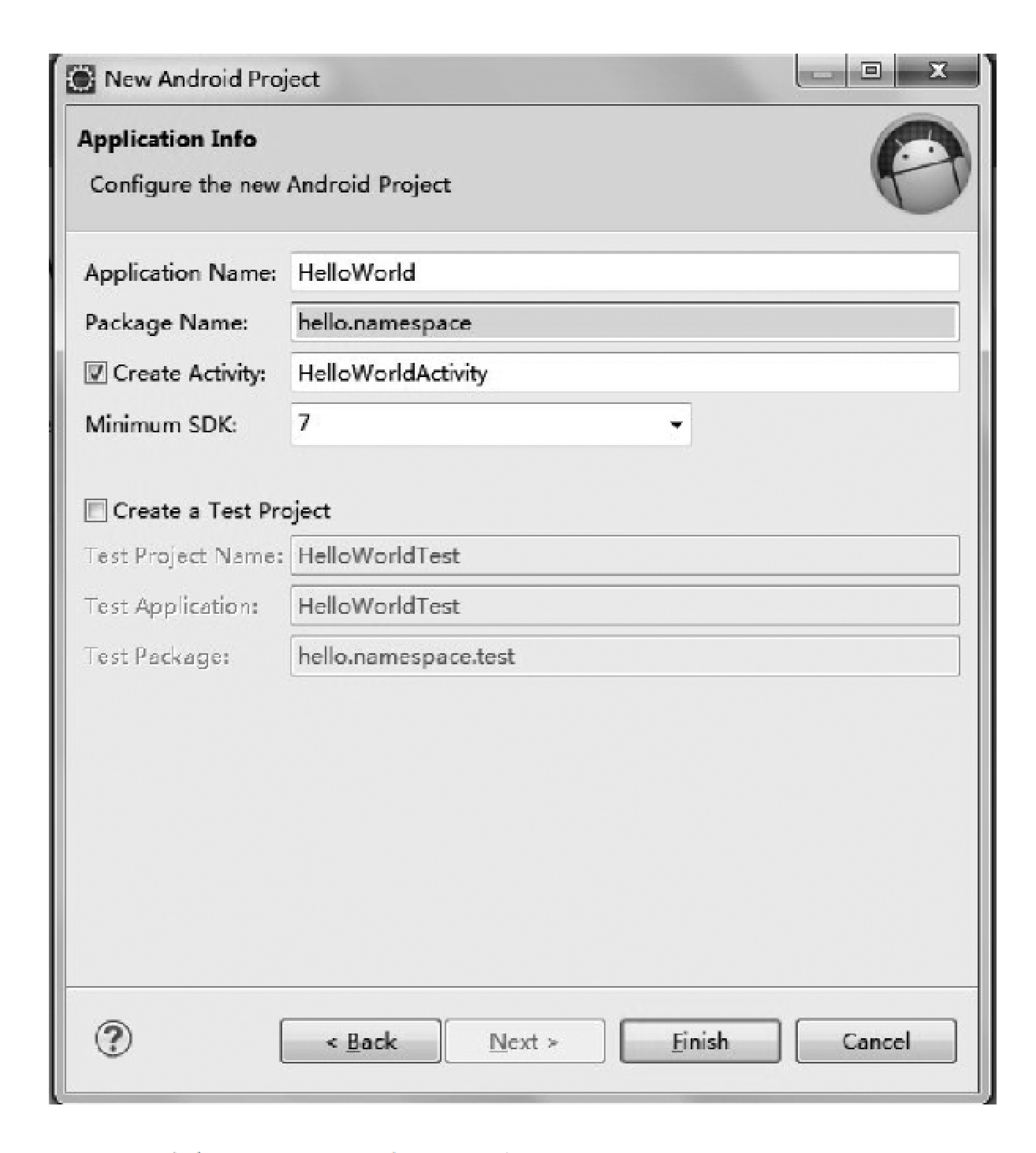


图 5-40　Configure the new Android Project

(5) 单击 Finish 按钮,完成 Android Project HelloWorld 创建。这时的 Eclipse 界面如图 5-41 所示。

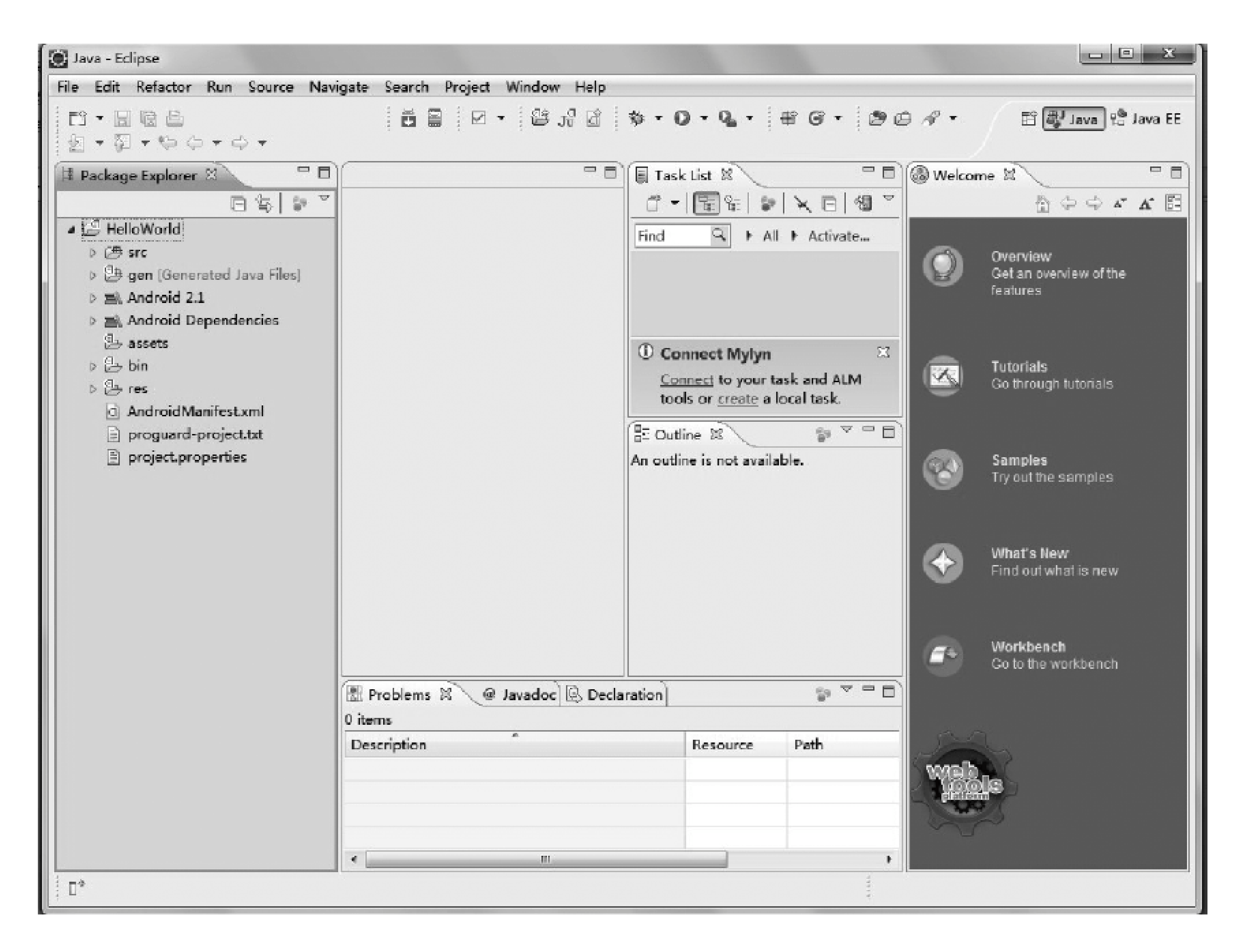


图 5-41　创建 HelloWorld 项目的 Java-Eclipse 界面

2）运行并测试 HelloWorld

（1）单击 Eclipse 的 Run→Run Configurations 菜单命令，出现 Run Configurations 界面，如图 5-42 所示。

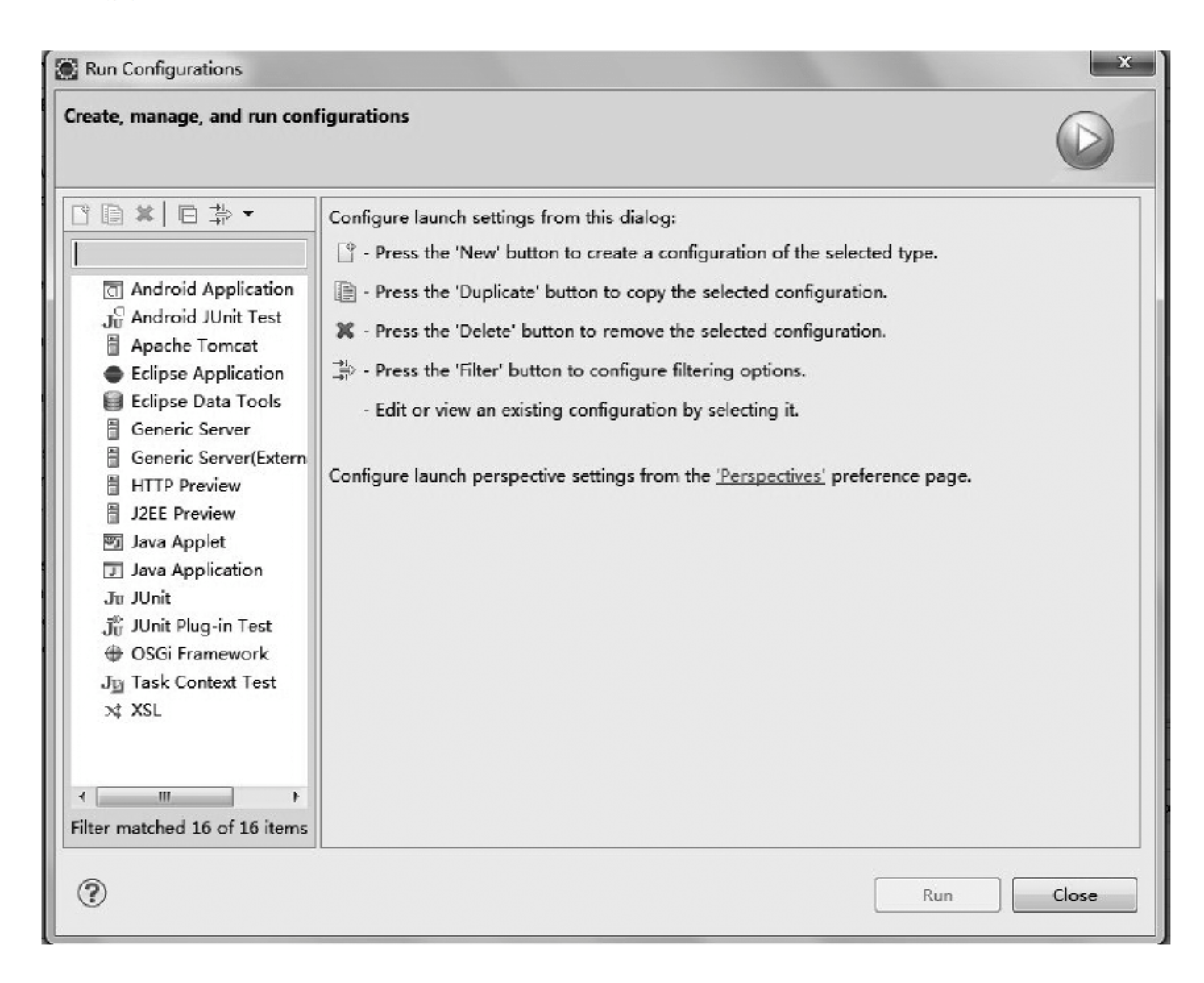


图 5-42　Run Configurations 界面

（2）选择 Android Application 项，单击左上角的图标按钮 或者双击 Android Application，在其下方出现一个新的选项 New_configuration，这里将其 Name 改为 hello，如图 5-43 所示。

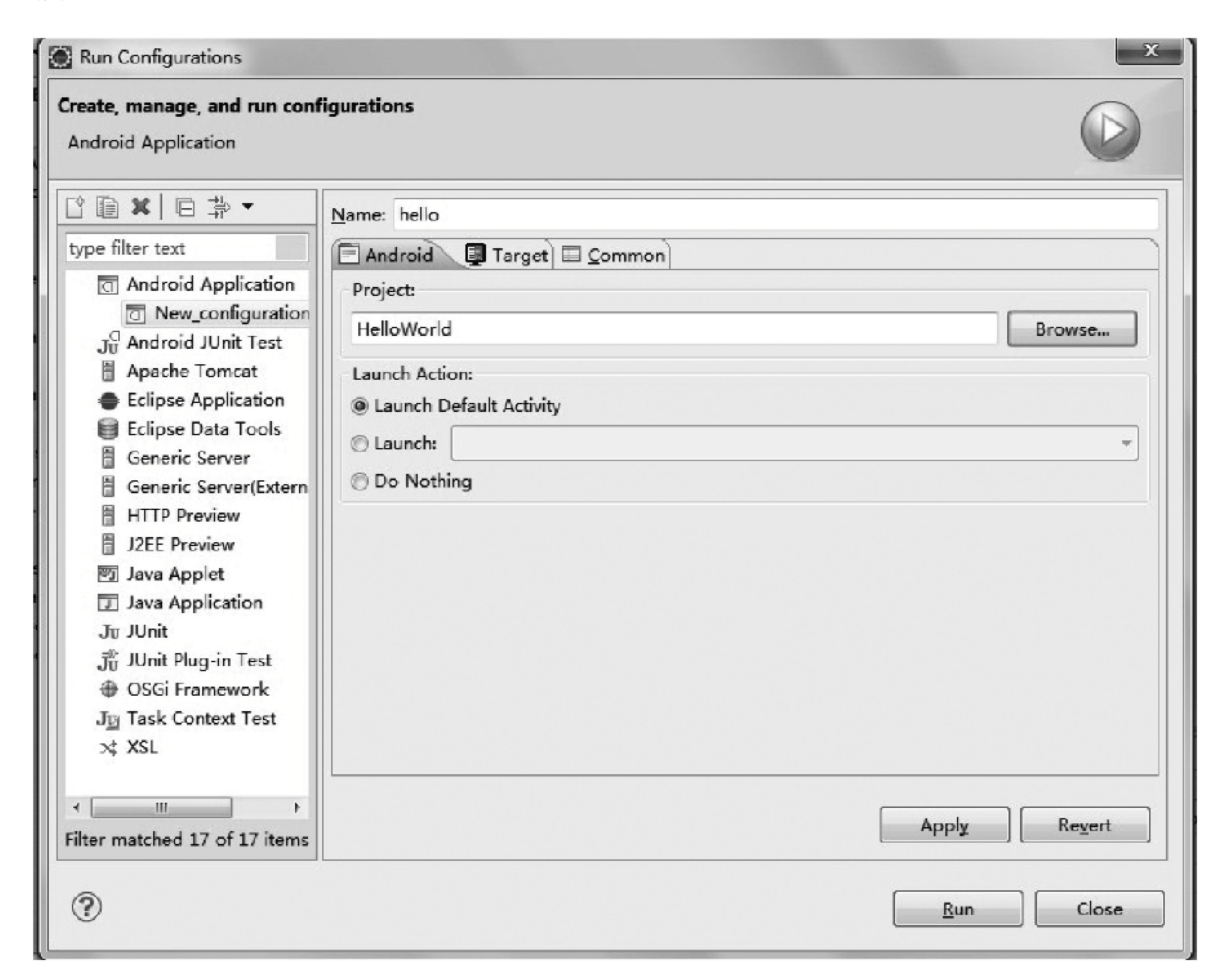


图 5-43 Android Application

（3）在右侧 Android 选项卡中单击 Browse 按钮，在弹出的 Project Selection 对话框中选择 HelloWorld 项，如图 5-44 所示。

（4）在图 5-43 中的 Launch Action 选项区域选择 Launch Default Activity 单选按钮，即建立默认的活动。

（5）在图 5-43 中打开 Target 选项卡，选中 Automatic 单选按钮并选中相应的 AVD 复选框。如果没有可用的 AVD 则单击右下角的 Manager 按钮，然后新建相应的 AVD，如图 5-45 所示。

（6）单击 Apply 按钮后，可以看到 Apply 按钮变成灰显，如图 5-46 所示。

（7）单击 Run 按钮运行 HelloWorld 程序。若运行成功会有 Android 的 Emulator 界面，如图 5-47 所示。

（8）测试程序运行结果，如图 5-48 所示。

（9）单击右边栏的 图标按钮可以进入 Android Emulator 的主界面，如图 5-49 所示。

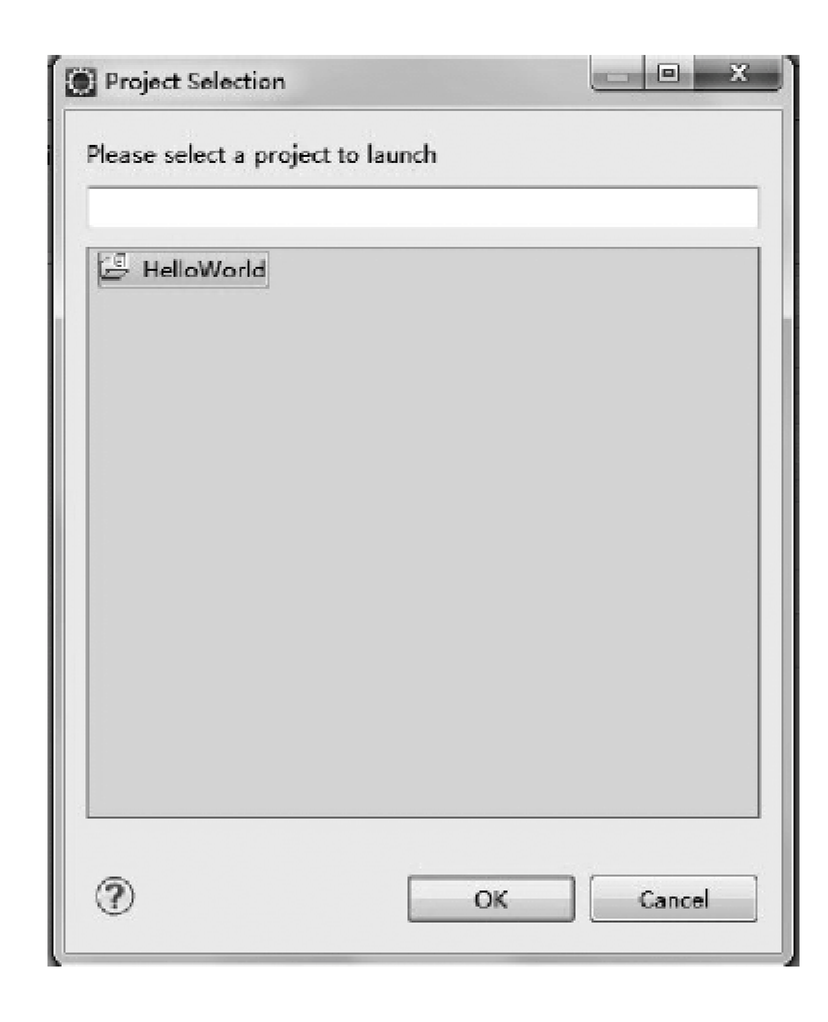


图 5-44 Project Selection 对话框

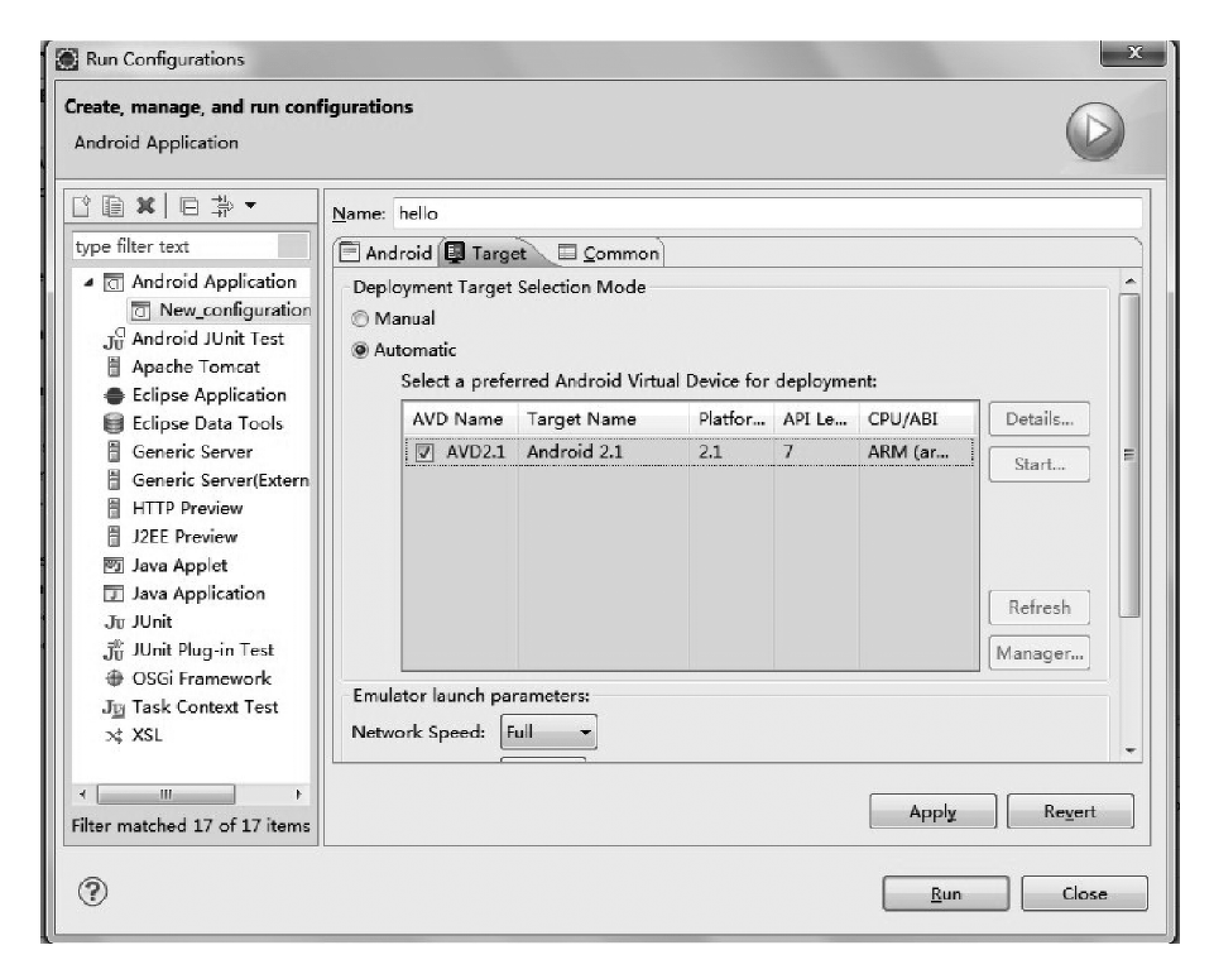


图 5-45　Android Application-Target

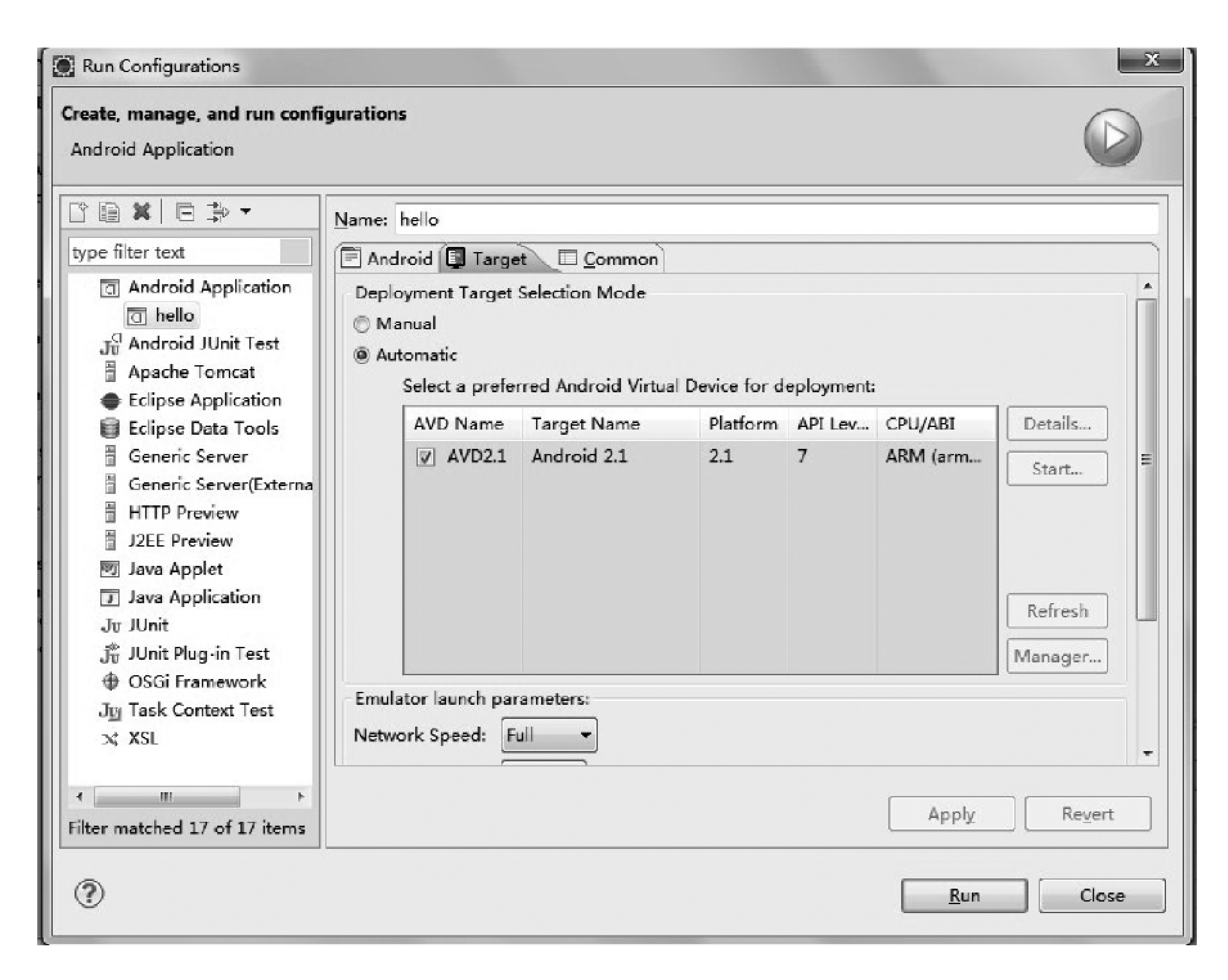


图 5-46　Android Application-Apply

如图 5-50 所示。

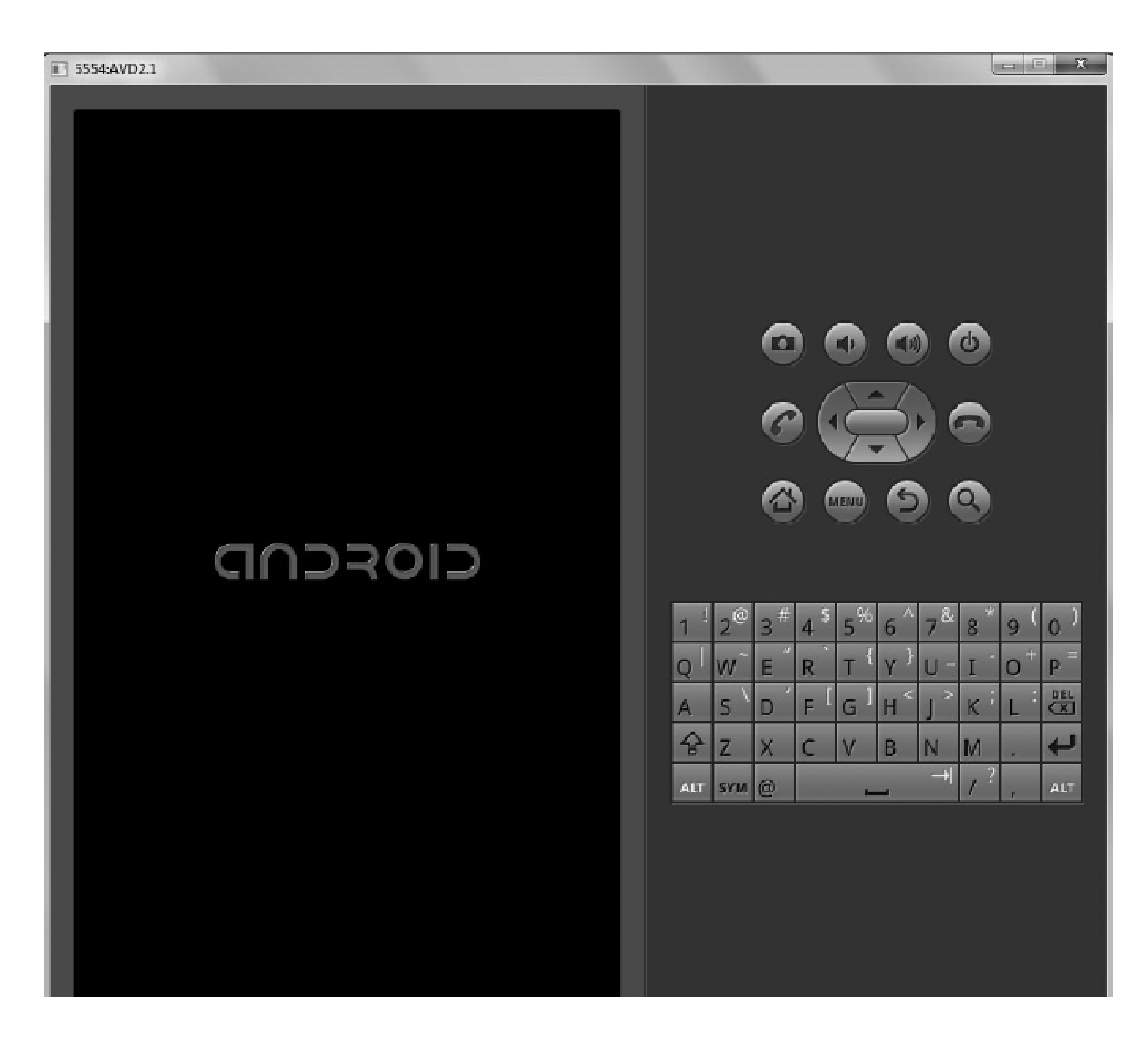


图 5-47　Android Emulator 进入界面

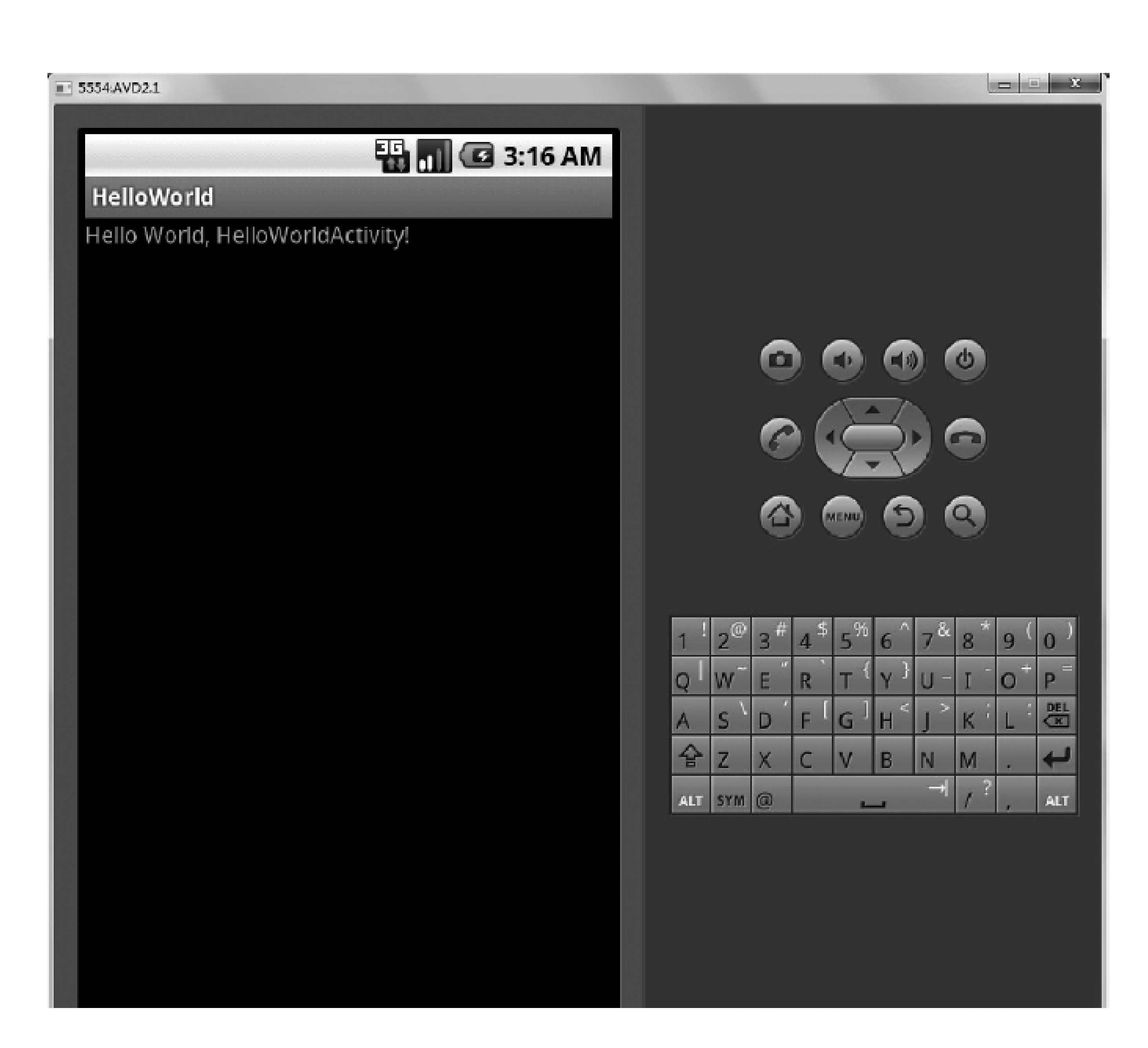


图 5-48　测试结果

(10) 单击 Android Emulator 主界面最下角的图标按钮，就进入安装程序界面，如图 5-50 所示。

至此，Windows 7 下的 Android 整个开发环境已经搭建完成。Windows XP 系统下的 Android 开发环境的整个搭建过程与此基本相似。



图 5-49　Android Emulator 的主界面



图 5-50　Android 安装程序界面

5.1.2　Linux(Ubuntu)下 Android 开发环境搭建

1. 准备工作

Linux(Ubuntu)下安装 Android 开发环境的准备工作和 Windows 7 下的比较相似，区别在于 JDK 和 Eclipse 要下载 Linux 版本的，否则就没法安装了。

2. JDK 配置

这里以 JDK 在/usr/locallib/目录下为例。如果想把自己的 JDK 也放到该目录下，那么使用 sudo cp 命令复制即可。

1）修改权限

在 Shell 下执行命令：

```
sudo chmod +x jdk-6u20-linux-i586.bin
```

修改完权限后使用 ls 命令查看会发现 JDK 文件名变成绿色，这就可以了，如图 5-51 所示。

```
flysnow@ubuntu:~$ cd /usr/local/lib
flysnow@ubuntu:/usr/local/lib$ ls
jdk1.6.0_20  jdk-6u20-linux-i586.bin  python2.6
flysnow@ubuntu:/usr/local/lib$ sudo chmod +x jdk-6u20-linux-i586.bin
[sudo] password for flysnow:
flysnow@ubuntu:/usr/local/lib$ ls
jdk1.6.0_20  jdk-6u20-linux-i586.bin  python2.6
flysnow@ubuntu:/usr/local/lib$
```

图 5-51 修改 Linux 权限

2）安装

执行命令：

```
sudo ./jdk-6u20-linux-i586.bin
```

就开始安装 JDK 了，遇到协议和同意协议的时候回车或者输入“Y”回车即可。

3）配置环境变量

安装好后就可以开始配置环境变量了。执行命令：

```
sudo gedit /etc/profile
```

打开配置文件，在文件尾部加入以下文本：

```
JAVA_HOME=/usr/local/lib/jdk1.6.0_20
JRE_HOME=/usr/local/lib/jdk1.6.0_31/jre
CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib
PATH=$PATH:$JAVA_HOME/bin:/home/flysnow/bin
export PATH JAVA_HOME JRE_HOME CLASSPATH
```

保存关闭。但是这个时候输入“javac”还是会报错的，因为这个配置必须重启才能生效。重启后输入“java -version”就可以看到版本信息了，如图 5-52 所示。

4）设置默认 JDK

JDK 安装完成后，有的时候输入“java-version”显示的并不是 Java HotSpot(TM)Client VM，而是其他诸如 Open JDK 等，这是因为机器里默认安装的有其他版本的 JDK。以下操

```
flysnow@ubuntu:/usr/local/lib$ java -version
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode, sharing)
flysnow@ubuntu:/usr/local/lib$
```

图 5-52　Java 版本信息

作可以将自己的 JDK 设置成默认的。

（1）执行以下命令：

```
update-alternatives --install /usr/bin/java java /usr/local/lib/jdk1.6.0_20/bin/java 500
update-alternatives --install /usr/bin/javac javac /usr/local/lib/jdk1.6.0_20/bin/javac 500
```

（2）上面的命令是将刚安装的 JDK 加入 Java 选单。然后接着执行：

```
update-alternatives --config java
```

这是为系统选择默认的 JDK，也就是刚刚安装的 JDK。这时候再执行 Java -version 命令就会看到使用的是自己的 JDK 了。

3. Android SDK 配置

SDK 的配置和 JDK 的大同小异，比 JDK 的配置更加简单。这里以 SDK 目录在～/Dev/Frame/Android 下为例进行配置。

首先已经解压了 SDK 目录，然后使用 sudo gedit /etc/profile 命令打开配置文件，加入 Android 后的配置文件应该是如下这样的：

```
JAVA_HOME=/usr/local/lib/jdk1.6.0_20
ANDROID_HOME=/home/flysnow/Dev/Frame/Android/android-sdk
JRE_HOME=/usr/local/lib/jdk1.6.0_20/jre
CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib
PATH=$PATH:$JAVA_HOME/bin:/home/flysnow/bin:$ANDROID_HOME/tools
export PATH JAVA_HOME ANDROID_HOME JRE_HOME CLASSPATH
```

读者们可以参考上述代码修改自己的 SDK 配置文件，然后保存重启，SDK 就配置好了。

4. 其他

Eclipse 的安装、ADT 的安装、SDK 的 API、DOC 的下载这些和 Windows 7 下一样，可以参考 Windows 7 下搭建 Android 开发环境的内容。Linux 下和 Windows 7 下的配置步骤基本一样，都是下载软件、配置环境变量等。Linux 下的难点还是对于 shell 命令的掌握以及对于环境变量的配置，也即对于 Linux 系统掌握的程度，是否能熟练使用，这就是 Linux 的基本功问题了。

5.2 Android 应用程序的结构

5.2.1 Android 的开发环境

Android 开发环境有两种安装方式，即在线安装和离线安装，这在前文已进行过介绍。Android 应用程序开发过程中还应注意仿真器的使用、多仿真器的通信以及 Android Emulator 的应用。

1. Android Emulator 的功能

(1) 可模拟电话本、通话等功能。

(2) 内置的浏览器和 Google Maps 都可以联网。

(3) 可以使用键盘输入。

(4) 可单击 Emulator 按键输入。

(5) 可以使用单击、拖动屏幕进行操作。

2. Emulator 和真机的不同之处

(1) 不支持呼叫和接听实际来电，但可以通过控制台模拟电话呼叫(呼入和呼出)。

(2) 不支持 USB 连接。

(3) 不支持相机/视频捕捉。

(4) 不支持音频输入(捕捉)，但支持输出(重放)。

(5) 不支持扩展耳机。

(6) 不能确定连接状态。

(7) 不能确定电池电量水平和交流充电状态。

(8) 不能确定 SD 卡的插入/弹出。

(9) 不支持蓝牙。

需要注意的是，有时会遇到系统关于 C 盘空间不足之类的提示，这是由于 Android Emulator 每次运行时会临时生成几个. tmp 后缀的临时文件所致。一段时间后，其可能占用几 GB 的磁盘空间，需要手工定期清理。

5.2.2 Android 应用程序的构成

Android 系统没有使用常见的应用程序入口点的方法(例如 main())，应用程序就是由组件组成的，组件是可以调用的相互独立的基本功能模块。根据完成的功能不同，Android 划分了四类核心组件：Activity、Service、BroadcastReceiver 和 ContentProvider。

并不是每个应用程序都必须包含这 4 类组件，有的程序可能只包含部分组件，组件之间的导航通过 Intent 来完成。Android 应用程序的构成示意图如图 5-53 所示。

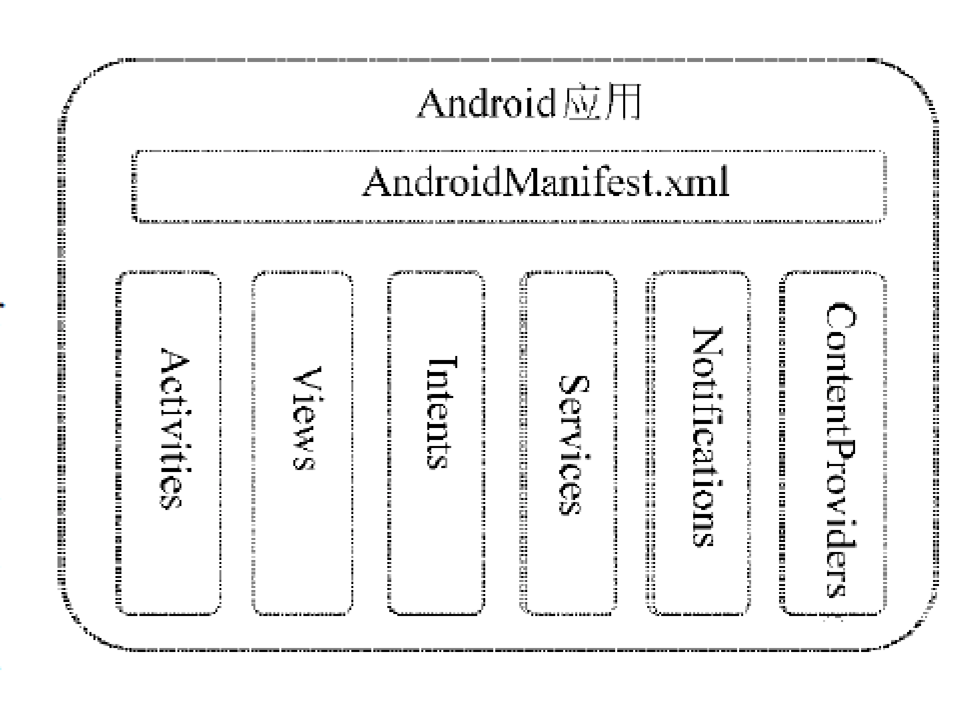


图 5-53 Android 应用程序的构成示意图

1．Activity

Activity 是 Android 程序的表示层。应用程序中的每个屏幕显示都通过继承和扩展基类 Activity 来实现，显示可视化的用户界面，并接收与用户交互所产生的界面事件。Activity 利用 View 来实现应用程序的 GUI（图形用户界面），而手机用户则直接通过 GUI 和应用程序做交互，例如应用程序通过 GUI 向用户显示信息，用户通过 GUI 向应用程序发出指令和响应。

Android 应用程序可以包含一个或多个 Activity，一般在程序启动后会呈现一个 Activity，用于提示用户程序已经正常启动。Activity 在界面上的表现形式分为全屏窗体、非全屏悬浮窗体和对话框。例如一个短信应用程序，需要一个 Activity 来显示联系人列表，同时需要另一个 Activity 显示用户输入的短信内容，甚至还可能需要第三个 Activity 显示已收到的短信内容。虽然这些 Activity 整体形成了一个完整的短信程序用户界面，但实际上每个 Activity 是独立的。当然，它们也有共同点，即每个 Activity 都是继承 Activity 的子类，如图 5-54 所示。

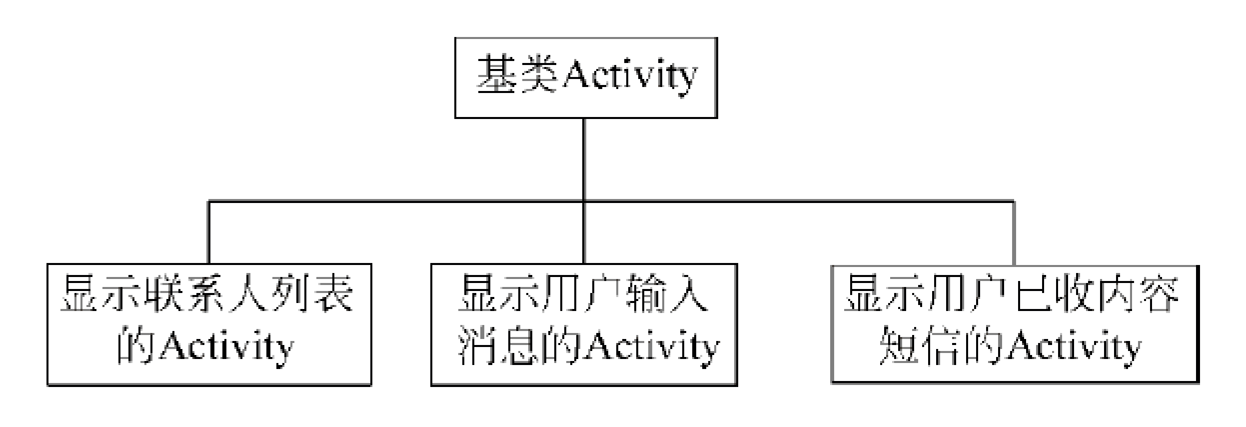


图 5-54　Activity 是表示层中的基类

2．Intents

Intents 负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述，Android 则根据此 Intents 的描述，负责找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用。

因此，Intents 在这里起着一个媒体中介的作用，专门提供组件互相调用的相关信息，实现调用者与被调用者之间的解耦。

例如，在一个联系人维护的应用中，当我们在一个联系人列表屏幕（假设对应的 Activity 为 listActivity）上，单击某个联系人后，希望能够跳出此联系人的详细信息屏幕（假设对应的 Activity 为 detailActivity）。为了实现这个目的，listActivity 需要构造一个 Intent，这个 Intent 用于告诉系统，我们要做“查看”动作，此动作对应的查看对象是“某联系人”，然后调用 startActivity (Intent intent)，将构造的 Intent 传入，系统会根据此 Intent 中的描述，到 ManiFest 中找到满足此 Intent 要求的 Activity，系统会调用找到的 Activity，即为 detailActivity，最终传入 Intent，detailActivity 则会根据此 Intent 中的描述，执行相应的操作。

3．Service

与 Activity 相反，Service 没有可见的用户界面，但 Service 的特点是能长时间在后台运行。因此也可以这样理解，Service 是具有一段较长生命周期且没有用户界面的程序。

4．BroadcastReceiver

BroadcastReceiver 是用来接收并响应广播消息的组件。广播是一种同时通知多个对象的事件通知机制。Andriod 中的广播通知来自系统或普通应用程序。很多事件都可能导

致系统广播，例如手机所在的时区发生变化、电池电量低、用户改变系统语言设置等；当然也有广播来自应用程序，例如一个应用程序通知其他应用程序某些数据已经下载完毕。

BroadcastReceiver 的特点：

(1) 不包含任何用户界面。

(2) 可以通过启动 Activity 或者 Notification 通知用户接收到重要信息。

(3) Notification 能够通过多种方法提示用户，包括闪动背景灯、震动设备、发出声音或在状态栏上放置一个持久的图标。

5. ContentProvider

ContentProvider 是 Android 系统提供的一种标准共享数据机制，应用程序可以通过 ContentProvider 访问其他应用程序的私有数据。私有数据可以是存储在文件系统中的文件，也可以是 SQLite 中的数据库。

Android 系统内部也提供一些内置的 ContentProvider，能够为应用程序提供重要的数据信息。根目录包含的数据如图 5-55 所示，共有 4 个子目录，即 src、assets、res 和 gen；一个库文件 android.jar；两个工程文件，即 AndroidManifest.xml 和 default.properties。

1) src 目录

src 目录是源代码目录，所有允许用户修改的 java 文件和用户自己添加的 java 文件都保存在这个目录中。

2) gen 目录

gen 目录用来保存 ADT 自动生成的 java 文件。例如 R.java，包含对 drawable、layout 和 values 目录内的资源的引用指针，Android 程序能够直接通过 R 类引用目录中的资源。

(1) R.java 文件不能手工修改，如果向资源目录中增加或删除了资源文件，则需要在项目名称上右击，选择 Refresh 来更新 R.java 文件中的代码。

(2) R 类包含的几个内部类分别与资源类型相对应，资源 ID 便保存在这些内部类中，例如子类 drawable 表示图像资源，内部的静态变量 icon 表示资源名称，其资源 ID 为 0x7f020000。一般情况下，资源名称与资源文件名相同。

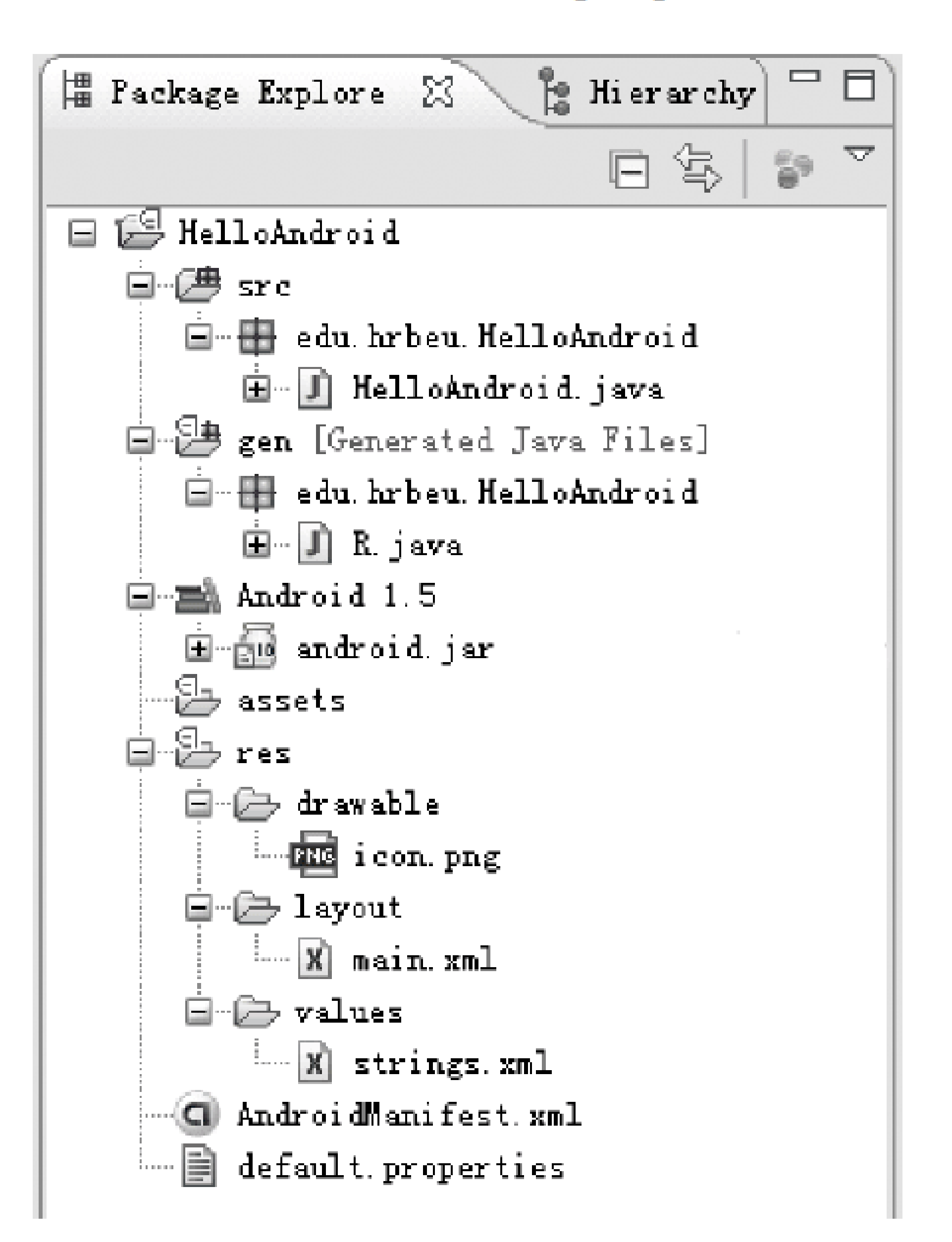


图 5-55　ContentProvider

(3) 资源引用有两种情况：一种是在代码中引用资源；另一种是在资源中引用资源。代码中引用资源需要使用资源的 ID，可以通过[R.resource_type.resource_name]或[android.R.resource_type.resource_name]获取资源 ID。其中：

resource_type 代表资源类型，也就是 R 类中的内部类名称；

resource_name 代表资源名称，对应资源的文件名或在 XML 文件中定义的资源名称属性。

资源中引用资源的引用格式为：@ [package:]type:name。其中：

@表示对资源的引用;

package 是包名称。如果在相同的包则 package 可以省略。

3) android. jar 文件

android. jar 文件是 Android 程序所能引用的函数库文件,Android 通过平台所支持的 API 都包含在这个文件中。

4) assets 目录

assets 目录用来存放原始格式的文件,例如音频文件、视频文件等二进制格式文件。此目录中的资源不能被 R. java 文件索引,所以只能以字节流的形式读取,一般情况下为空。

5) res 目录

res 目录是资源目录,有 3 个子目录用来保存 Android 程序的所有资源。

(1) drawable 目录用来保存图像文件。

(2) layout 目录用来保存与用户界面相关的布局文件。

(3) values 目录保存文件颜色、风格、主题和字符串等。

(4) 在 HelloAndroid 项目中,ADT 在 drawable 目录中自动引入了 icon. png 文件作为 HelloAndroid 程序的图标文件,在 layout 目录生成了 main. xml 文件用于描述用户界面。

(5) main. xml 文件是界面布局文件,利用 XML 语言描述用户界面。

(6) strings. xml 文件用来存放定义的字符串。

(7) AndroidManifest. xml 文件

AndroidManifest. xml 文件是 XML 格式的 Android 程序声明文件,包含了 Android 系统运行 Android 程序前所必须掌握的重要信息,这些信息包含应用程序名称、图标、包名称、模块组成、授权和 SDK 最低版本等,而且每个 Android 程序必须在根目录下包含一个 AndroidManifest. xml 文件。

(1) AndroidManifest. xml 文件的代码如下:

```
<?xml version = "1.0" encoding = "utf - 8"?>
<manifest xmlns:android = "http://schemas.android.com/apk/res/android"
        package = "edu.hrbeu.HelloAndroid"
        android:versionCode = "1"
        android:versionName = "1.0">
    <application android:icon = "@drawable/icon"
                    android:label = "@string/app_name">
        <activity android:name = ".HelloAndroid"
                    android:label = "@string/app_name">
            <intent - filter>
                <action android:name = "android.intent.action.MAIN" />
                <category android:name = "android.intent.category.LAUNCHER" />
            </intent - filter>
        </activity>
    </application>
    <uses - sdk android:minSdkVersion = "3" />
</manifest>
```

(2) Android Manifest. xml 文件的主要功能如下:

① 指定了该应用程序的 Java 包:该包名作为应用程序的一个独特标识。

② 描述了应用程序组件：该应用程序由哪些 Activity、Service、BroadcastReceiver 和 ContentProvider 组成。

③ 指定了实现每个组件的类以及公开发布它们的能力(例如它们能持有哪个 Intent 信息)。这些声明使 Android 系统知道这里有什么组件以及在什么条件下它们可以被载入。

④ 决定哪些进程将容纳应用程序组件。

⑤ 声明了本应用程序必须拥有哪些许可,以便访问 API 的被保护部分以及与其他应用程序交互。也声明了其他应用程序在和该应用程序交互时需要持有的许可。

⑥ 列出了 Instrumentation 类,可以在应用程序运行时提供简档和其他信息。这些声明仅当应用程序在开发和测试过程中被提供;将在应用程序正式发布之前被移除。

⑦ 声明了该应用程序所需的 Android API 的最小化水平。

⑧ 列出了该应用程序必须链接的库。

(3) AndriodManifest. xml 的代码解释如下:

① <? xml version="1.0" encoding="utf-8"? > //清空内存,程序初始化,判断运行条件是否已准备好。

② <manifest> //根节点,描述了 package 中所有的内容。

③ <uses-permission /> //请求此 package 正常运行所需赋予的安全许可。一个 manifest 能包含零个或更多此元素。

④ <permission /> //声明了安全许可来限制哪些程序能使用此 package 中的组件和功能。一个 manifest 能包含零个或更多此元素:<permission-tree />;<permission-group />。

⑤ <instrumentation /> //声明了用来测试此 package 或其他 package 指令组件的代码。一个 manifest 能包含零个或更多此元素。

⑥ <uses-sdk /> //指定当前应用程序兼容的最低 SDK 版本号。

⑦ <application> //包含 package 中 application 级别组件声明的根节点。此元素也可包含 application 中全局和默认的属性,例如标签、icon、主题以及必要的权限等。一个 manifest 能包含零个或一个此元素(不允许多余一个)。

⑧ <activity> //用来与用户交互的主要工具。当用户打开一个应用程序的初始页面是一个 activity,大部分被使用到的其他页面也由不同的 activity 所实现并声明在另外的 activity 标签中。

⑨ <intent-filter> //声明了指定的一组组件支持的 Intent 值:

```
<action />
<category />
<data />
    <type/>
    <schema/>
        <authority/>
        <path/>
 </intent-filter>
  <meta-data />
 </activity>
```

```
<activity-alias>
  <intent-filter> ... </intent-filter>
  <meta-data />
 </activity-alias>
 <service>      //Service 是能在后台运行任意时间的组件
  <intent-filter> ... </intent-filter>
  <meta-data/>
 </service>
 <receiver>//
```

⑩ IntentReceiver 能使此 application 获得数据的改变或者发生的操作，即使它当前不在运行：

```
<intent-filter> ... </intent-filter>
<meta-data />
</receiver>
```

⑪ <provider> //ContentProvider 是用来管理持久化数据并发布给其他应用程序使用的组件。

```
      <grant-uri-permission />
      <meta-data />
    </provider>
    <uses-library />
    <uses-configuration />
</application>
</manifest>
```

(4) 在 AndriodManifest. xml 中声明权限。权限声明在智能手机中是一种比较常见的应用程序保护机制，其基本思想是通过 AndriodManifest. xml 文件中显式地声明应用程序需要的权限，防止应用程序错误地使用服务、不恰当地访问资源，最终达到提高 Andriod 应用程序的健壮性、改善用户体验的目的。Andriod 中每种权限都用一个独立的标签表示，例如：

```
andriod.permission.SEND_SMS
android.permission.CALL_PHONE
andriod.permission.READ_OWNER_DATA
```

从上述这些表示权限名称的标签中不难发现其所代表的含义，例如 andriod. permission. SEND_SMS 表示发送短信的权限。Andriod 系统的每种功能或每种特性都可以用一个权限来表示。自然地，如果应用程序希望访问某种特性，那么就需要在 AndriodManifest. xml 文件中对权限进行声明，否则应用程序将无法使用希望的功能，而且不会出现关于缺少权限的错误信息提示。因此开发人员应该特别留意权限，尤其是当应用程序莫名其妙不能正常工作时。

6) default. properties 文件

默认属性文件。在这个文件中，可以设置自己的代码属性，以使代码安全可靠，其他人

在不知道代码属性的情况下无法使用代码。

5.3 Android的虚拟机和Java环境

5.3.1 Dalvik虚拟机和核心库

Android的Java基本运行环境包括虚拟机和核心库两个方面，虚拟机是Java运行的基础，核心库提供尽量与标准.Java SE兼容的类库。

Dalvik是Android程序的虚拟机，是Android中Java程序的运行基础。Dalvik虚拟机主要完成对象生命周期管理、堆栈管理、线程管理、安全和异常管理以及垃圾回收等重要功能。Dalvik虚拟机是基于寄存器的，所有的类都经由Java编译器编译，Aradroid中的dx工具将其转化成dex格式由虚拟机执行。

Dalvik虚拟机相关的内容在Android中是一个独立的代码路径dalvik/，其中包含了目标机和主机的内容。

1. dex工具库和虚拟机的实现

dalvik/dex目录中的内容是dex工具库；dalvik/vm目录中的内容是虚拟机的实现库；dalvik/dalvikvn目录中的内容是虚拟机的可执行程序。dex工具库最终将生成静态库libdex.a；Dalvik虚拟机的实现库将生成libdvm.so（Dalvik虚拟机）。libdvm.so连接静态库libdex.a；虚拟机的可执行程序dalvik将连接libdvm.so。

与标准的Java虚拟机的字节码不同，Dalvik虚拟机所用的字节码是dex格式（Dalvik Executable）。Dalvik的可执行文件针对小内存使用做了优化。dex格式内容生成的最终文件是dex文件（对应于标准Java的class文件）。dex字节码和标准Java的字节码（Class）在结构上的一个区别是dex字节码将多个文件整合成一个，dex工具库的作用就是生成和处理dex文件的工具集合。这是一个由纯C语言写成的工具库。

通用的虚拟机是基于堆栈的，Dalvik虚拟机是基于寄存器的。这样做在通用性和可移植性上差了一些，但是却可以获得更好的性能。Dalvik虚拟机的底层由C语言和汇编语言实现，它还依赖Linux内核的一些功能，例如线程机制和底层内存管理机制。

Dalvik虚拟机的实现库libdvm.so是虚拟机的核心内容，它的大体内容由纯C语言实现，但是为了在关键环节实现优化，某些部分是特定体系结构的汇编语言以及与体系结构相关的C语言。其Android.mk的片段如下所示：

```
Ifeq( $ (TARGET_ARCR),arm)
   LOCAL_SRC_FILES += \
        Arch/arm/CallOldABI.S\
        Arch/arm/CallEABI.S\
        Arch/arm/HintsEABI.C
LOCAL_SRC_FILES += \
    Mterp/out/InterpC - armv5te.c.arm\
    Mterp/out/InterpAsm - armv5te.s\
    LOCAL_SHARED_LIBRARUWS += Libdl
Else
```

```
    Ifeq( $ (TARGET_ARCH), * 86)
      LOCAL_SRC_FILES += \
        Arch/ * 86/Call386ABI.S\
        Arch/ * 86/Hints386ABI.c
      LOCAL_SRC_FILES += \
        Mterp/out/InterpC - x86.c\
        Mterp/out/InterPAsm - x86.s
Else
     LOCAL_C_INCLUDES += external/libffi/ $ (TARGET_OS) - s(TARGET_ARCH)
      LOCAL_SRC_FILES += \
        Arch/generic/Call.c\
        Arch/generic/Hints.c
      LOCAL_SHARED_LIBRARIES += libffi
      LOCAL_SRC_FILES += \
        Mterp/out/Interpc - allstubs.c\
        Mterp/out/InterpcAsm - allstubs.s\
```

其中主要的区别代码在于 arch(体系结构相关目录)和 mterp(Dalvik 解释器目录),根据宏 TARGET_ARCH(目标机体系结构)做出选择:

(1) 如果目标机是 ARM 体系结构,则使用 ARM 的汇编语言及其相关的 C 语言,默认为 ARMv5TE 体系结构,也可以使用 ARMv4 体系结构。

(2) 如果目标机是×86 体系结构,则使用×86 的汇编语言及其相关的 C 语言。

(3) 如果目标机是其他体系结构,则使用通用的 C 语言实现,有一部分代码是与体系结构无关的汇编语言实现。

2. 核心库

核心库(libcore)提供了基本的 Java 类库的功能,包含了基础数据结构、数学函数、IO、工具、数据库、网络等多方面的内容。

libcore 目录中包含了 C 语言、C++和 Java 的实现,也包含了部分测试程序,主要的编译结果为:

(1) libjavacore.a:本地代码经过编译后将生成本地代码的 Java 核心静态库。主要代码来自 luni、luni-kernel、icu、archive、sql 等目录的本地代码部分。

(2) Java 包 Core.jar:经过安装后这个包最终将被放置在目标文件系统的/system/framework 目录中。core.jar 本身是 java 程序使用的一个包,解压缩后其主要结构如图 5-56 所示。

其中,classes.dex 为主体的 Dalvik 可执行格式的字节码,java 和 org 目录下分别是 java 和 org 这两个包中某些类包含的一些属性。

3. nativehelper 库

nativehelper 库是一个工具库,用于注册 Java 本地调用的函数,在其他代码中需要使用 JNI 从本地层次向 Java 层次提供功能时需要使用这个库。

nativehelper 库的代码路径为 dalvik/libnativehelper,连接静态库 libjavacore.a,生成动态库 libnativehelper.so。

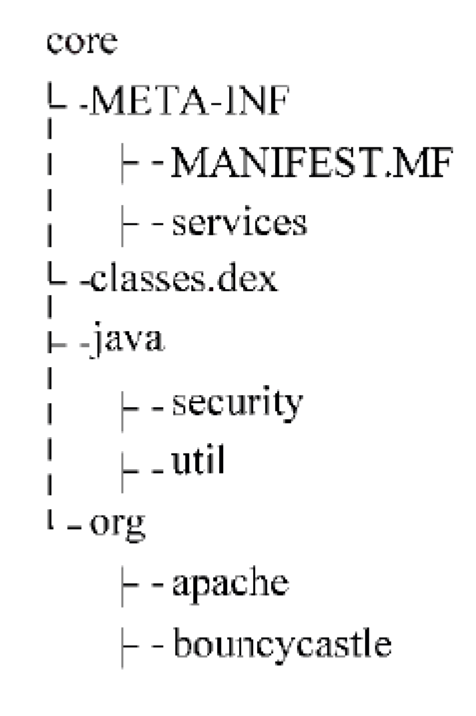


图 5-56 core.jar 主要结构

nativehelper 库的头文件路径为：

(1) libnativehelper/include/nativehelper/jni.h：基于 JNI 标准头文件。

(2) libnativehelper/include/nativehelper/JNIHelp.h：提供 JNI 注册功能头文件。

在其他程序实现 JNI 时需要包含这两个头文件。

5.3.2 Android 的 Java 程序环境

1. Java 类的层次结构

在 Android 系统的层次结构中，自下而上的第 3 个层次是 Java 程序的框架，第 4 个层次是 Java 应用程序，在它们之间的接口是 Android 系统 Java API。这是 Android 系统的基本 API，其层次结构主要包含以下部分：

(1) Java 标准 API(Java 包)。

(2) Java 扩展 API(Javax 包)。

(3) 企业和组织提供的 Java 类库。

(4) Android 的各种包(android 包)。

其中，前 3 个部分都基本是 Java 语言中的标准部分，在 Android 中具有与标准 Java 名称和功能相同的 API 功能。这些包在 Dalvik 虚拟机的目录 dalvik/libcore 中进行了实现。Android API 的说明可以参考 Android 帮助文档中的 Reference 部分，其中具有按照类和按照包的说明。

值得注意的是，Android 对标准 Java 的支持并不是一个全集，某些标准 Java 的 API 在 Android 系统中没有支持。其中比较典型的是，由于 Android 具有自己的 GUI 系统，因此标准 Java 和 GUI 相关的部分在 Android 中没有支持，例如 Swing 和 AWT。

Android 的各种包以 android 开头，这个包是 Android 中提供的主要 API，独立于标准 Java API。

2. Android Java 类的代码

Android 的应用程序基础是各个 Android 提供的各个 Java 类，这些类就是 Android 系统架构中第 3 层的内容——Java 框架，其代码主要在 frameworks/base/目录中。其中主体部分将生成 frameworks.jar 包，最终将被放置在目标文件系统的/system/framework 目录中。

Android 中的 Java 库主要是 android 包及其子包，其中核心包的目录为 Frameworks/base/core/java/。

除了上述核心目录之外，在 frameworks 目录中还有几个子目录包含了 Java 框架部分的代码，例如：

(1) frameworks/base/graphics/java/：图形部分的 Java 类。

(2) frameworks/base/media/java/：媒体部分的 Java 类。

(3) frameworks/base/opengl/java/：OPenGL 部分的 Java 类。

(4) frameworks/base/wifi/java/：部分的 Java 类。

凡是 Java 框架部分的内容，其目录和代码的组织形式都是类似的，即各个子目录和文件是按照 Java 包的关系来组织的，文件夹的层次结构表示包和子包，文件名称和 Java 类的名称一致。例如文件 android/app/Activity.java，它表示 android.app 包中的类 Activity，也就是类 android.app.Activity。

此外，Android 的 Java 系统中存在共有资源文件的概念，例如可显示的图片、多语言字符串、布局文件、XML 文件、纯数据文件等。一些内部资源文件也被生成一个 jar 包，其源路径为 Frameworks/base/core/res/。这些内容最后会生成一个名称为 framework-res. apk 的包，这个包也被放置到目标文件系统的/system/frameworks 目录中。

3. Android 系统 API

Android 适用于各种各样的手机，从最低端直到最高端的智能手机。核心的 Android API 在每部手机上都可使用，但仍然有一些 API 接口有一些特别的适用范围：这就是所谓的"可选 API"。

这些 API 之所以是"可选的"，主要是因为一个手持设备并不一定要完全支持这类 API，甚至于完全不支持。例如，一个手持设备可能没有 GPS 或 Wi-Fi 的硬件。在这个条件下，这类功能的 API 依然存在，但不会以相同的方式来工作。

应用应该无障碍地运行或连接在一个可能不支持 API 的设备，因为设备上有这些上层接口（the classes）。当然执行起来可能什么也不做，或者抛出一个异常。

Android 系统对提供的 API 具有严格的组织形式，因为这些 API 兼容性的基础，如前面所述，Android 系统的 API 是 Android 的 Java 的接口，使 Android 系统保持 Java 框架层和 Java 应用层。

（1）Android 中 Java 类的 API 描述文件包含在 frameworks/base/api/目录中，主要是其中的 current. xml 文件。current. xml 文件的所有内容都包含在＜api＞标签中。current. xml 内部的内容组织是以包（package）分类的。例如，对于 android. app 包中的类 Activity，其代码路径为 android/app/Activity. java。代码片段如下所示：

```
Public class Activity extends ContextThemeWrapper     //定义 Activity 类
      Implements LayoutInflater.Factory,
      Window.Callback,KeyEvent.Callback,
      OnCreateContextMenuListener,ComponentCallbacks{
    Public Activity(){
          ++sInstanceCount;
}
//省略 Activity 类的内容
}
```

提示：在 Java 源代码中，必须定义与文件名同名。

这个源文件对应的 current. xml 中的内容片段如下所示：

```
<package name> = "android.app">
<class name = "Activity"
Extends = "android.view.ContextThemeWrapper"
Abstract = "false"
Static = "false"
Final = "false"
  Deprecated = "not deprecated"
Visibility = "public"
>
<implements name = "android.content.ComponentCallbacks">
```

```
</implements>
<implements name="android.view.ReyEvent. Callback">
</implements>
<implements name="android.view.LayoutInflater.Factory">
</implements>
<implements name="android.view.View.OnCreateContextMenuListener">
</implements>
<implements name="android.view.Window.Callback">
</implements>
<constructor name="Activity"
Type="Android.app.Activity"
Static="false"
Final="false"
Deprecated="not deprecated"
Visibility="public"
>
</constructor>
<!-- 省略内容 -->
</class><-- 省略内容 -->
</package>
```

(2) Android 中 JAVA 类的 API 的描述文件包含在 frameworks/base/api/目录的 current.xml 文件。主要使用的标签如下:

```
<package> </package>
<class> </class>
<interface> </interface>                /*接口*/
<implements> </implements>              /*实现接口*/
<method> </method>                      /*方法*/
<field> </field>                        /*字段、域*/
```

当注释中写入@hide 的时候,就表示内容被隐藏了,即这个内容虽然出现在 JAVA 的源代码中,但是不被视为属于 Android 的系统 API。

① 包的内容:整个 android.app 包的内容都包含在<package name="android.app">和</package>标签中。

② 类的内容:由于要实现对类 Activity 的定义,因此使用了<class name="Activity">和</class>标签,并指定了这个类所继承的类,在这里继承了类 android.view。ContextThemeWrapper 其他的几个值如 abstract、static、final 等表示这个类的一些特性。

③ 类实现的内容:描述 Activity 所实现的几个接口,每个接口用一对<implements>和</implements>标签。

④ 构造函数的内容:Activity 构造函数的内容用一对<constructor>和</constructor>来描述,名称 name 肯定是 Activity,返回值 type 也肯定是类的类型 android.app.Activity,这个构造函数没有任何参数,所以没有参数的定义。

android.app.Activity 类的定义:

```
public class Activity extends ContextThemeWrapper // 定义 Activity 类
implements LayoutInflater.Factory,
```

```
Window.Callback, KeyEvent.Callback,
OnCreateContextMenuListener, ComponentCallbacks {
public Activity() {
++sInstanceCount;
}
//省略
}
<package name = "android.app">
<class name = "Activity"
extends = "android.view.ContextThemeWrapper"
abstract = "false"
static = "false"
final = "false"
deprecated = "not deprecated"
visibility = "public">
<implements name = "android.content.ComponentCallbacks"></implements>
<implements name = "android.view.KeyEvent.Callback"></implements>
<implements name = "android.view.LayoutInflater.Factory"></implements>
<implements name = "android.view.View.OnCreateContextMenuListener"></implements>
<implements name = "android.view.Window.Callback"></implements>
<constructor name = "Activity"
type = "android.app.Activity"
static = "false"
final = "false"
deprecated = "not deprecated"
visibility = "public"
>
</constructor>
<!-- 省略内容 -->
</class>
<!-- 省略内容 -->
</package>
```

接口和类的情况类似，区别只是包含在<interface>和</interface>标签中。此外，接口中的 abstract 值通常为 true，表示接口是抽象的，需要继承才能够使用。

值得注意的是，代码中的注释会影响和 API 描述文件的对应关系。尤其是当注释中写入@hide 时，就表示内容被隐藏了，即这个内容虽然出现在 Java 源代码中，但是不被视为属于 Android 的系统 API。

5.3.3 JNI 的使用

JNI 是 Java Native Interface 的缩写，中文为“Java 本地接口”。从 Java 1.1 开始，JNI 标准成为 Java 平台的一部分，它允许 java 代码和用其他语言编写的代码进行交互。JNI 是本地编程接口，它使得在 java 虚拟机(VM)内部运行的 Java 代码能够与用其他编程语言(如 C、C++和汇编语言)编写的应用程序和库进行交互操作。

JNI 是 JAVA 标准平台中的一个重要功能，它弥补了 JAVA 的与平台无关这一重大优点的不足，在 JAVA 实现跨平台的同时，也能与其他语言(如 C、C++)的动态库进行交互，给

其他语言发挥优势的机会。有了 JAVA 标准平台的支持，使 JNI 模式更加易于实现和使用。

1. JNI 的架构和实现方式

在 Android 中提供 JNI 的方式，让 Java 程序可以调用 C 语言。Android 中很多 Java 类都具有 native 代接口，这些 native 接口就是由本地实现，然后注册到系统中的。

在 Android 中，主要的 JNI 代码在以下的路径中：

```
Frameworks/base/core/jni/
```

这个路径中的内容将被编译成库 libandroid_runtime. so，这就是一个普通的动态库，被放置在目标系统的/system/lib 目录中。

除此之外，Android 还包含其他的 JNI 库，例如，媒体部分的 JNI 在目录 frameworks/base/media/jni 中，被编译成库 libmedia_ini. so。

JNI 中的各个文件实际上就是 C++的普通源文件，其命名一般和支持的 Java 类有对应关系。这种关系是习惯上的写法，而不是强制的。

在 Android 中实现的 JNI 库，需要连接动态库 libnativehelper. so。

提示：Android 中使用 JNI 时，尤其需要注意 JNI 函数参数和返回值的类型。

1）在框架层实现 JNI

android. util. Log 类的情况：

```
static JNINativeMethod gMethods[] = {
{ "isLoggable", "(Ljava/lang/String;I)Z",
(void * ) android_util_Log_isLoggable },
{ "println","(ILjava/lang/String;Ljava/lang/String;)I",
(void * ) android_util_Log_println },
};
public final class Log {
public static native boolean isLoggable(String tag, int level);
public static native int println(int priority, String tag, String msg);
}
```

2）android_util_Log. cpp 中的方法列表

注册 JNI 的情况：

```
int register_android_util_Log(JNIEnv * env)
{
jclass clazz = env->FindClass("android/util/Log");
//省略其他处理的内容
levels.verbose = env->GetStaticIntField(clazz,
env->GetStaticFieldID(clazz, "VERBOSE", "I"));
levels.debug = env->GetStaticIntField(clazz,
env->GetStaticFieldID(clazz, "DEBUG", "I"));
levels.info = env->GetStaticIntField(clazz,
```

```
env->GetStaticFieldID(clazz, "INFO", "I"));
levels.warn = env->GetStaticIntField(clazz,
env->GetStaticFieldID(clazz, "WARN", "I"));
levels.error = env->GetStaticIntField(clazz,
env->GetStaticFieldID(clazz, "ERROR", "I"));
levels.assert = env->GetStaticIntField(clazz,
env->GetStaticFieldID(clazz, "ASSERT", "I"));
return AndroidRuntime::registerNativeMethods(env, "android/util/Log",
gMethods, NELEM(gMethods));
}
```

3）在 Apk 中实现 JNI

JNI 的示例程序的路径：

(1) development/samples/SimpleJNI。

(2) 编译成的 JNI 动态库：libsimplejni.so。

(3) 编译成的 Java 包：SimpleJNI.apk。

路径代码如下所示：

```
TOP_LOCAL_PATH := $(call my-dir)
# Build activity
LOCAL_PATH := $(TOP_LOCAL_PATH)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := samples
LOCAL_SRC_FILES := $(call all-subdir-java-files)
LOCAL_PACKAGE_NAME := SimpleJNI
LOCAL_JNI_SHARED_LIBRARIES := libsimplejni
include $(BUILD_PACKAGE)
include $(call all-makefiles-under, $(LOCAL_PATH))
```

4）JNI 中的高级使用

JNI 的基本功能是让 Java 调用本地代码。除此之外，JNI 还可以实现更为高级的内容。

(1) JNI 代码如下所示：

```
struct fields_t {
jfieldID context;
jmethodID post_event;
};
static fields_t fields;
/* 获取和调用 */
{
fields.context = env->GetFieldID(clazz, "mNativeContext", "I");
env->SetIntField(thiz, fields.context, (int)context);
fields.post_event = env->GetStaticMethodID(clazz,
"postEventFromNative",
"(Ljava/lang/Object;IIILjava/lang/Object;)V");
}
```

（2）与之对应的 Java 代码：

```
public class TestClass {
@SuppressWarnings("unused")
private int mNativeContext;
private EventHandler mEventHandler;
TestClass() {
/* ...... */
test(new WeakReference<TestClass>(this));
}
private class EventHandler extends Handler
{
private TestClass mTestClass;
public EventHandler(TestClass testclass, Looper looper) {
super(looper);
mTestClass = testclass;
}
@Override
public void handleMessage(Message msg) {
msg.what;
msg.arg1;
msg.arg2;
/* ...... */
}
}
private static void postEventFromNative(Object TestClassref,
int what, int arg1, int arg2, Object obj)
{
TestClass TestClass = (TestClass)((WeakReference)TestClassref).get();
if (TestClass == null)
return;
if (TestClass.mEventHandler != null) {
Message m = TestClass.mEventHandler.obtainMessage(what, arg1, arg2, obj);
TestClass.mEventHandler.sendMessage(m);
}
}
}
```

5）系统服务的 Java 部分

Java 层同样提供了一套 Binder 的相关函数，让 Java 代码可以直接进行 Binder 操作。实现在：

```
frameworks/base/core/java/android/os/
frameworks/base/core/java/com/android/internal/os/
frameworks/base/core/jni/
```

2. 在应用程序中使用 JNI

根据上节介绍，JNI 可以提供给 Java 框架使用的代码，相当于定义在了 Android 的系统 API 层次。如果只需要在一个程序中通过 JNI 调用本地程序，那么不需要更改 Android 的系统 API 也可以按照本节介绍的方法实现。

SimpleJNl 提供了在应用程序中通过 JNI 调用本地程序，可以作为参考，代码的路径为：development/samples/SimpleJNI。

5.4 Android 用户界面开发

5.4.1 用户界面基础

用户界面(User Interface，UI)是系统和用户之间进行信息交换的媒介，实现信息的内部形式与人们可以接受形式之间的转换。

在计算机出现早期，批处理界面(1945—1968)和命令行界面(1969—1983)得到广泛的使用。目前普遍使用图像用户界面(Graphical User Interface，GUI)，即采用图形方式与用户进行交互的界面。未来的用户界面将更多地运用虚拟现实技术，使用户能够摆脱键盘与鼠标的交互方式，而通过动作、语言甚至是脑电波来控制计算机。

1. 手机用户界面

设计手机用户界面应解决的问题是界面设计与程序逻辑完全分离，这样不仅有利于其并行开发，而且在后期修改界面时，也不用再次修改程序的逻辑代码；根据不同型号手机的屏幕解析度、尺寸和纵横比自动调整界面上部分控件的位置和尺寸，避免因为屏幕信息的变化而出现显示错误，能够合理利用较小的屏幕显示空间构造出符合人机交互规律的用户界面，避免出现凌乱、拥挤的用户界面。

Android 已经解决了上述两个问题：使用 XML 文件描述用户界面；资源文件独立保存在资源文件夹中；对用户界面描述非常灵活，允许不明确定义界面元素的位置和尺寸，仅声明界面元素的相对位置和粗略尺寸。

2. Android 用户界面框架

Android 用户界面框架(Android UI Framework)采用 MVC(Model-View-Controller)模型，提供了处理用户输入的控制器(Controller)、显示用户界面和图像的视图(View)以及保存数据和代码的模型(Model)，如图 5-57 所示。

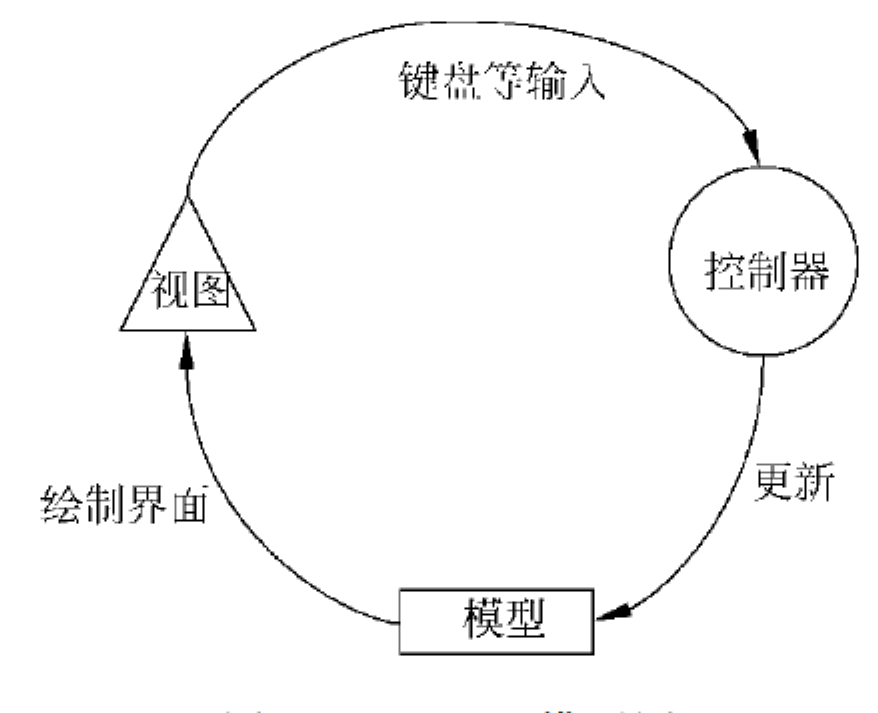


图 5-57 MVC 模型图

1) MVC 模型

MVC 模型中的控制器能够接收并响应程序的外部动作，例如按键动作或触摸屏动作等，控制器使用队列处理外部动作，每个外部动作作为一个独立的事件被加入队列中，然后 Android 用户界面框架按照“先进先出”的规则从队列中获取事件，并将这个事件分配给所对应的事件处理函数。

2) 视图树

Android 用户界面框架采用视图树(View Tree)模型，即其界面元素以一种树型结构组织在一起，称为视图树。Android 系统会依据视图树的结构从上至下绘制每一个界面元素，每个元素负责对自身的绘制，如果元素包含子元素则该元素会通知其下所有子元素进行绘制，如图 5-58 所示。

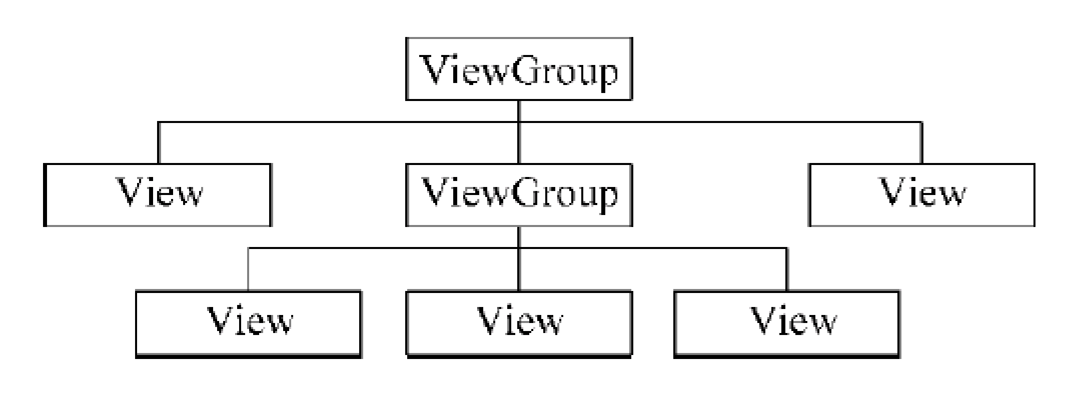


图 5-58　视图树的组成结构

视图树由 View 和 ViewGroup 构成。View 是界面的最基本可视单元，存储了屏幕上特定矩形区域内所显示内容的数据结构，并能够实现所占据区域的界面绘制、焦点变化、用户输入和界面事件处理等功能；View 也是一个重要的基类，所有在界面上的可见元素都是 View 的子类。ViewGroup 是一种能够承载含多个 View 的显示单元，其功能是承载界面布局和承载具有原子特性的重构模块。

3. 单线程用户界面

在单线程用户界面中，控制器从队列中获取事件和视图在屏幕上绘制用户界面，使用的都是同一个线程。其特点是处理函数具有顺序性，能够降低应用程序的复杂程度，同时也能降低开发的难度。缺点是如果事件处理函数过于复杂，可能会导致用户界面失去响应。

5.4.2　界面控件

Android 系统的界面控件分为定制控件和系统控件。定制控件是用户独立开发的控件，或通过继承并修改系统控件后所产生的新控件，能够为用户提供特殊的功能或与众不同的显示需求方式。系统控件是 Android 系统提供给用户已经封装的界面控件，提供在应用程序开发过程中的常见功能控件。

系统控件更有利于帮助用户进行快速开发，同时能够使 Android 系统中应用程序的界面保持一致性。常见的系统控件包括 TextView、EditText、Button、ImageButton、Checkbox、RadioButton、Spinner、ListView 和 TabHost。

1. TextView 和 EditText

TextView 是一种用于显示字符串的控件。EditText 则是用来输入和编辑字符串的控件，它是一个具有编辑功能的 TextView。

建立一个名为 TextViewDemo 的程序，包含 TextView 和 EditText 两个控件，上方“用户名”部分使用的是 TextView，下方的文字输入框使用的是 EditText，如图 5-59 所示。

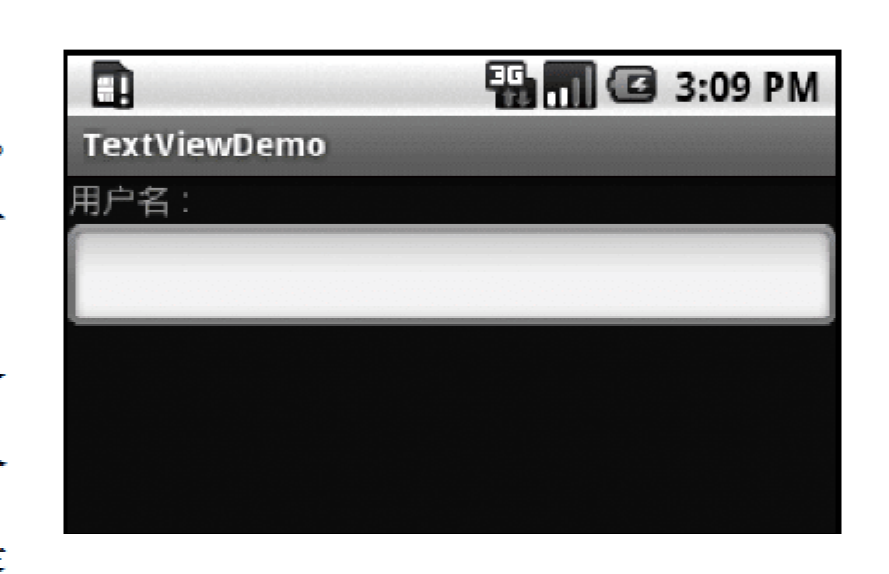


图 5-59　TextViewDemo 示意图

TextViewDemo 在 XML 文件中的代码如下：

```
<TextView android:id = "@ + id/TextView01"
android:layout_width = "wrap_content"
android:layout_height = "wrap_content"
android:text = "TextView01" >
</TextView>
```

```
6. <EditText android:id = "@ + id/EditText01"
7. android:layout_width = "fill_parent"
8. android:layout_height = "wrap_content"
9. android:text = "EditText01" >
10. </EditText>
```

(1) 第 1 行 android:id 属性声明了 TextView 的 ID,这个 ID 主要用于在代码中引用这个 TextView 对象。@+id/TextView01 表示所设置的 ID 值,@表示后面的字符串是 ID 资源;加号+表示需要建立的新资源名称,并添加到 R.java 文件中;斜杠后面的字符串 TextView01 表示新资源的名称;如果资源不是新添加的,或属于 Android 框架的 ID 资源,则不需要使用加号+,但必须添加 Android 包的命名空间,例如:android:id="@android:id/empty"。

(2) 第 2 行的 android:layout_width 属性用来设置 TextView 的宽度,wrap_content 表示 TextView 的宽度只要能够包含所显示的字符串即可。

(3) 第 3 行的 android:layout_height 属性用来设置 TextView 的高度。

(4) 第 4 行表示 TextView 所显示的字符串,在后文中将通过代码更改 TextView 的显示内容。

(5) 第 7 行中 fill_content 表示 EditText 的宽度将等于父控件的宽度。

TextViewDemo.java 文件中代码的修改如下:

```
1. TextView textView = (TextView)findViewById(R.id.TextView01);
2. EditText editText = (EditText)findViewById(R.id.EditText01);
3. textView.setText("用户名: ");
4. editText.setText("");
```

第 1 行代码的 findViewById()函数能够通过 ID 引用界面上的任何控件,只要该控件在 XML 文件中定义过 ID 即可。第 3 行代码的 setText()函数用来设置 TextView 所显示的内容。

2. Button 和 ImageButton

Button 是一种按钮控件,用户能够在该控件上单击并引发相应的事件处理函数。ImageButton 是用以实现能够显示图像功能的控件按钮。

建立一个名为 ButtonDemo 的程序,包含 Button 和 ImageButton 两个按钮,上方是 Button 按钮,下方是一个 ImageButton 控件,如图 5-60 所示。



图 5-60 ButtonDemo 示意图

1) ButtonDemo 在 XML 文件中的代码

```
<Button android:id = "@ + id/Button01"
 android:layout_width = "wrap_content"
 android:layout_height = "wrap_content"
 android:text = "Button01" >
</Button>
```

上述代码定义了 Button 控件的高度、宽度和内容；定义了 ImageButton 控件的高度和宽度，但是没定义显示的图像，在后面的代码中进行定义。

2）引入资源

将 download.png 文件复制到/res/drawable 文件夹下；在/res 目录上选择 Refresh，新添加的文件将显示在/res/drawable 文件夹下，R.java 文件内容也得到了更新；否则提示无法找到资源的错误。

3）更改 Button 和 ImageButton 内容

引入 android.widget.Button 和 android.widget.ImageButton：

```
1. Button button = (Button)findViewById(R.id.Button01);
2. ImageButton imageButton = (ImageButton)findViewById(R.id.ImageButton01);
3. button.setText("Button 按钮");
4. imageButton.setImageResource(R.drawable.download);
```

第 1 行代码用于引用在 XML 文件中定义的 Button 控件；第 2 行代码用于引用在 XML 文件中定义的 ImageButton 控件；第 3 行代码将 Button 的显示内容更改为“Button 按钮”；第 4 行代码利用 setImageResource()函数，将新加入的 png 文件 R.drawable.download 传递给 ImageButton。

4）按钮响应单击事件

添加单击事件的监听器，代码如下：

```
1. });
2. imageButton.setOnClickListener(new View.OnClickListener() {
3. public void onClick(View view) {
4.      textView.setText("ImageButton 按钮");
5. }
6. });
```

第 2 行代码中 image Button 对象通过调用 setOnClickListener()函数，注册一个单击(Click)事件的监听器 View.OnClickListener()；第 3 行代码是单击事件的回调函数；第 4 行代码将 TextView 的显示内容更改为“Button 按钮”。

5）View.OnClickListener()

View.OnClickListener()是 View 定义的单击事件的监听器接口，并在接口中仅定义了 onClick()函数。当 Button 从 Android 界面框架中接收到事件后，首先检查这个事件是否是单击事件，如果是单击事件，同时 Button 又注册了监听器，则会调用该监听器中的 onClick()函数。每个 View 仅可以注册一个单击事件的监听器，如果使用 setOnClickListener()函数注册第二个单击事件的监听器，之前注册的监听器将被自动注销。

多个按钮注册到同一个单击事件的监听器上，代码如下：

```
Button.OnClickListener buttonListener = new Button.OnClickListener(){
@Override
```

```
public void onClick(View v) {
    switch(v.getId()){
      case R.id.Button01:
        textView.setText("Button 按钮");
        return;
```

3. CheckBox 和 RadioButton

CheckBox 是一个同时可以选择多个选项的控件；RadioButton 则是仅可以选择一个选项的控件。RadioGroup 是 RadioButton 的承载体，程序运行时不可见，应用程序中可能包含一个或多个 RadioGroup；一个 RadioGroup 包含多个 RadioButton，在每个 RadioGroup 中，用户仅能够选择其中一个 RadioButton，如图 5-61 所示。

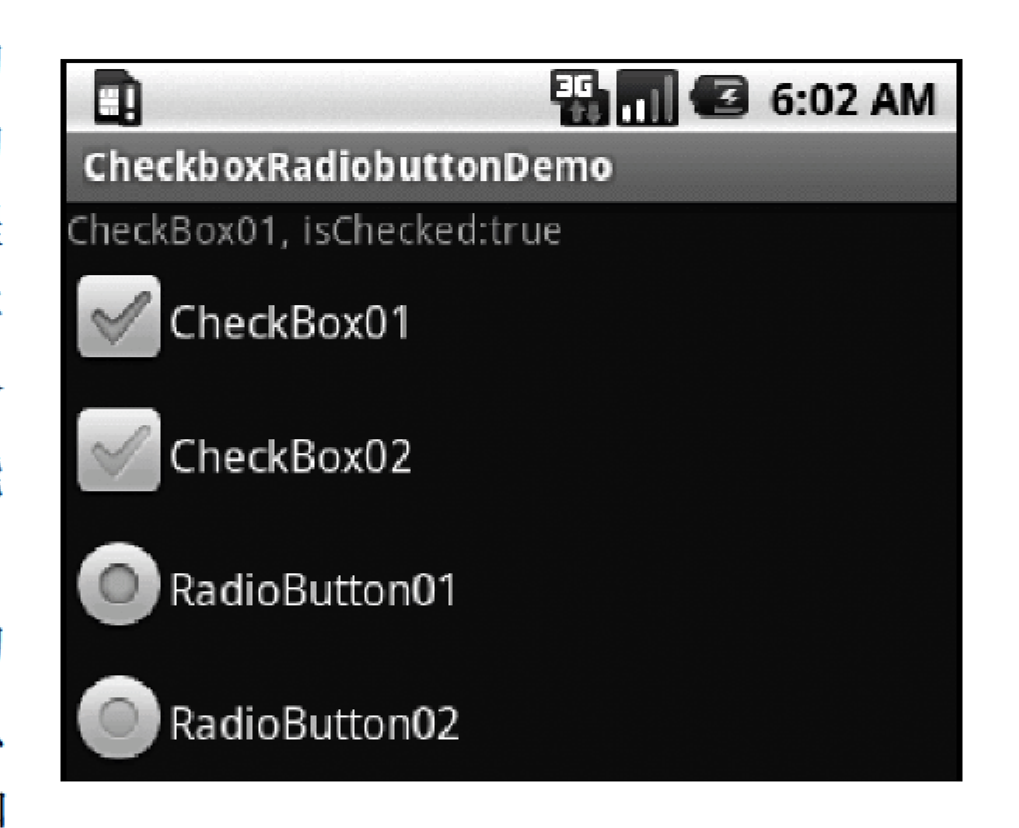


图 5-61　CheckboxRadiobuttonDemo 示意图

建立一个名为 CheckboxRadiobuttonDemo 的程序，包含 5 个控件，从上至下分别是 TextView01、CheckBox01、CheckBox02、RadioButton01 和 RadioButton02，如图 5-61 所示，当选择 RadioButton01 时，则 RadioButton02 无法选择。

(1) CheckboxRadiobuttonDemo 在 XML 文件中的代码如下：

```
<TextView android:id = "@ + id/TextView01"
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "@string/hello"/>
<CheckBox android:id = "@ + id/CheckBox01"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "CheckBox01" >
</CheckBox>
<CheckBox android:id = "@ + id/CheckBox02"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "CheckBox02" >
</CheckBox>
            <RadioGroup android:id = "@ + id/RadioGroup01"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content">
    <RadioButton android:id = "@ + id/RadioButton01"
      android:layout_width = "wrap_content"
      android:layout_height = "wrap_content"
      android:text = "RadioButton01" >
    </RadioButton>
```

```
23.     <RadioButton android:id = "@ + id/RadioButton02"
24.       android:layout_width = "wrap_content"
25.       android:layout_height = "wrap_content"
26.       android:text = "RadioButton02" >
27.     </RadioButton>
28.     </RadioGroup>
```

第 15 行<RadioGroup>标签声明了一个 RadioGroup；在第 18 行和第 23 行分别声明了两个 RadioButton，这两个 RadioButton 是 RadioGroup 的子元素。

（2）引用 CheckBox 和 RadioButton 的方法参考下面的代码：

```
CheckBox checkBox1 =  (CheckBox)findViewById(R.id.CheckBox01);
```

（3）CheckBox 设置单击事件监听器的简要代码如下：

```
CheckBox.OnClickListener checkboxListener  =  new CheckBox.OnClickListener(){
@Override
public void onClick(View v) {
  //过程代码
}};
checkBox1.setOnClickListener(checkboxListener);
checkBox2.setOnClickListener(checkboxListener);
```

与 Button 设置单击事件监听器中介绍的方法相似，唯一不同在于将 Button.OnClickListener 换成了 CheckBox.OnClickListener。

（4）RadioButton 设置单击事件监听器的方法如下：

```
RadioButton.OnClickListener radioButtonListener  =  new RadioButton.OnClickListener(){
@Override
public void onClick(View v) {
  //过程代码
}};
radioButton1.setOnClickListener(radioButtonListener);
radioButton2.setOnClickListener(radioButtonListener);
```

4. Spinner

Spinner 是一种能够从多个选项中选择一个选项的控件，类似于桌面程序的组合框（ComboBox），但没有组合框的下拉菜单，而是使用浮动菜单为用户提供选择。建立一个程序 SpinnerDemo，包含 3 个子项 Spinner 控件，如图 5-62 所示。

SpinnerDemo 在 XML 文件中的代码如下：

```
<TextView android:id = "@ + id/TextView01"
        android:layout_width = "fill_parent"
```

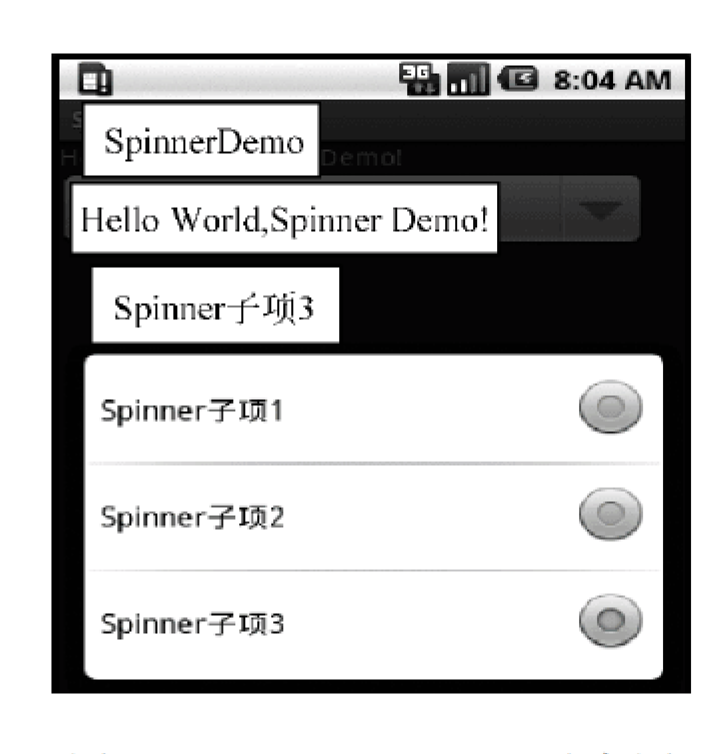


图 5-62　SpinnerDemo 示意图

```
3.         android:layout_height = "wrap_content"
4.         android:text = "@string/hello"/>
5.  < Spinner android:id = "@ + id/Spinner01"
6.    android:layout_width = "300dip"
7.    android:layout_height = "wrap_content">
8.  </Spinner >
```

第 5 行使用<Spinner>标签声明了一个 Spinner 控件；第 6 行代码中指定了该控件的宽度为 300dip。

在 SpinnerDemo. java 文件中，定义一个 ArrayAdapter 适配器，在 ArrayAdapter 中添加需要在 Spinner 中可以选择的内容，需要在代码中引入 android. widget. ArrayAdapter 和 android. widget. Spinner，即：

```
1. Spinner spinner  =  (Spinner) findViewById(R.id.Spinner01);
2. List < String > list  =  new ArrayList < String >();
3. list .add("Spinner 子项 1");
4. list .add("Spinner 子项 2");
5. list .add("Spinner 子项 3");
6. ArrayAdapter < String > adapter  =  new ArrayAdapter < String >(this,
     android.R.layout.simple_spinner_item, list );
7. adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
8. spinner.setAdapter(adapter);
```

第 2 行代码建立了一个字符串数组列表（ArrayList），这种数组列表可以根据需要进行增减；<String>表示数组列表中保存的是字符串类型的数据。

在代码的第 3、4、5 行中，使用 add()函数分别向数组列表中添加 3 个字符串。

第 6 行代码建立了一个 ArrayAdapter 的数组适配器，数组适配器能够将界面控件和底层数据绑定在一起。

第 7 行代码设定了 Spinner 的浮动菜单的显示方式，其中，android. R. layout. simple_spinner_dropdown_item 是 Android 系统内置的一种浮动菜单。

第 8 行代码实现绑定过程，所有 ArrayList 中的数据将显示在 Spinner 的浮动菜单中，设置 android. R. layout. simple_spinner_item 浮动菜单，显示结果如图 5-63 所示。

适配器绑定界面控件和底层数据，如果底层数据更改了，用户界面也相应修改显示内容，就不需要应用程序再监视，从而极大地简化了代码的复杂性。

5. ListView

ListView 是一种用于垂直显示的列表控件，如果显示内容过多，则会出现垂直滚动条。ListView 能够通过适配器将数据和自身绑定，在有限的屏幕上提供大量内容供用户选择，是经常使用的用户界面控件。ListView 支持单击事件处理，用户可以用少量的代码实现复杂的选择功能。

建立一个名为 ListViewDemo 的程序，包含 4 个控件，从上至下分别为 TextView01、ListView01、ListView02 和 ListView03，如图 5-64 所示。

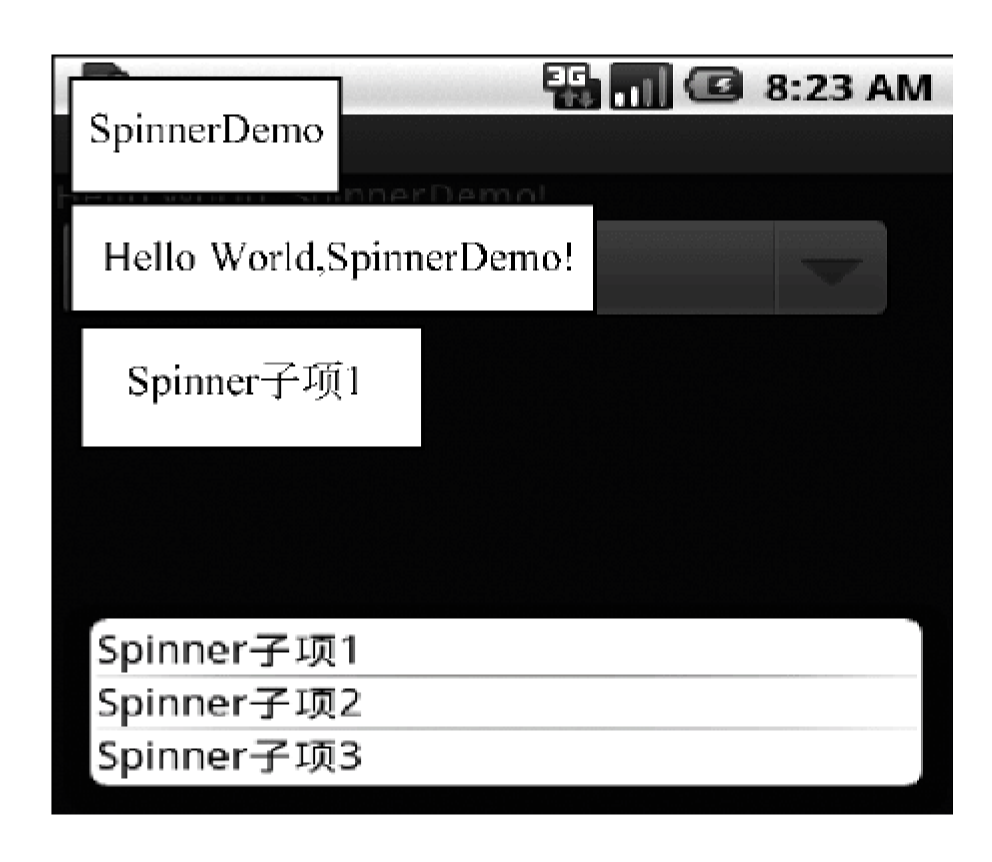


图 5-63　显示结果

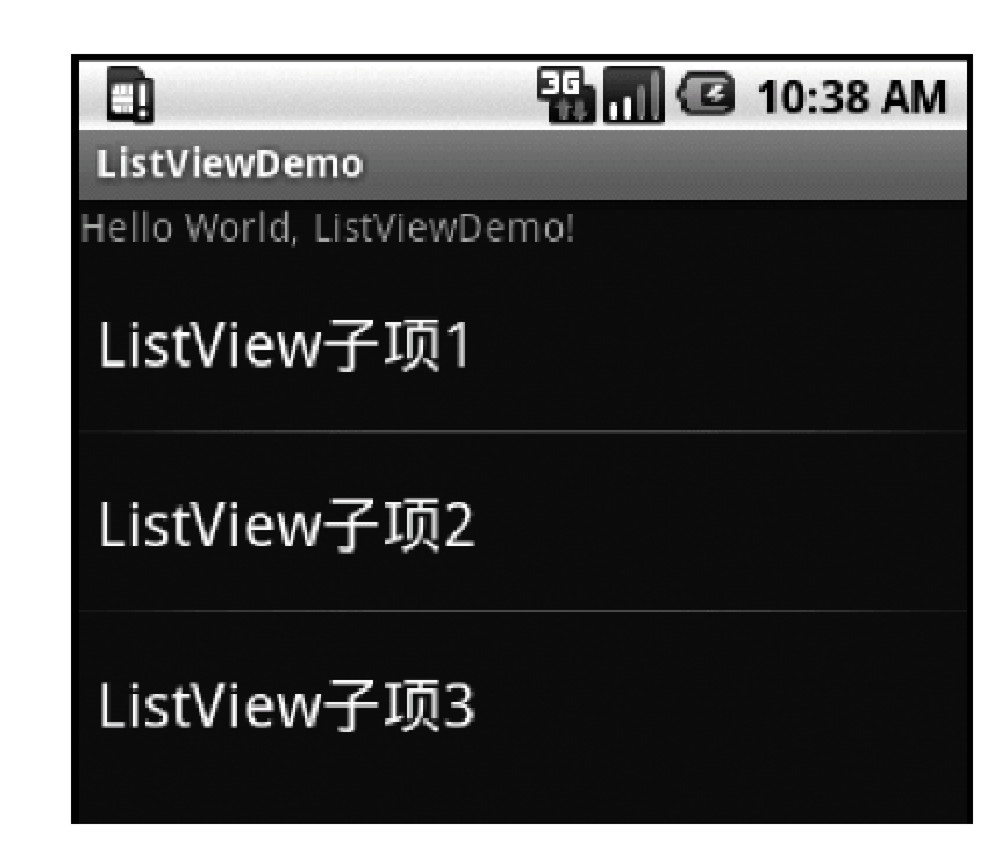


图 5-64　ListViewDemo 示意图

（1）ListViewDemo 在 XML 文件中的代码如下：

```
<TextView android:id = "@ + id/TextView01"
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "@string/hello" />
<ListView android:id = "@ + id/ListView01"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content">
</ListView>
```

（2）在 ListViewDemo.java 文件中，首先需要为 ListView 创建适配器，并添加 ListView 中所显示的内容，代码如下：

```
1. final TextView textView  =  (TextView)findViewById(R.id.TextView01);
2. ListView listView  =  (ListView)findViewById(R.id.ListView01);
3. List<String> list  =  new ArrayList<String>();
4. list.add("ListView 子项 1");
5. list.add("ListView 子项 2");
6. list.add("ListView 子项 3");
7. ArrayAdapter<String> adapter  =  new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, list );
8. listView.setAdapter(adapter);
```

第 2 行代码通过 ID 引用了 XML 文件中声明的 ListView；第 7 行代码声明了适配器 ArrayAdapter，第 3 个参数 list 说明适配器的数据源为数组列表；第 8 行代码将 ListView 和适配器绑定。

（3）下面的代码声明了 ListView 子项的单击事件监听器，用以确定用户在 ListView 中选择的是哪一个子项：

```
AdapterView.OnItemClickListener listViewListener  =  new
    AdapterView.OnItemClickListener(){
@Override
```

```
3. public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
4.     String msg = "";
5.     textView.setText(msg);
6. }};
7. listView.setOnItemClickListener(listViewListener);
```

第 1 行的 AdapterView.OnItemClickListener 是 ListView 子项的单击事件监听器，同样是一个接口，需要实现 onItemClick()函数。在 ListView 子项被选择后，onItemClick()函数将被调用。

第 3 行的 onItemClick()函数中一共有 4 个参数，参数 0 表示适配器控件，就是 ListView；参数 1 表示适配器内部的控件，是 ListView 中的子项；参数 2 表示适配器内部的控件，也就是子项的位置；参数 3 表示子项的行号。

第 4 行和第 5 行代码用于显示信息，选择子项确定后，在 TextView 中显示子项父控件的信息、子控件信息、位置信息和 ID 信息。

第 7 行代码是 ListView 指定刚刚声明的监听器。

6. TabHost

TabHost 即 Tab 标签页，是界面设计时经常使用的界面控件，可以实现多个分页之间的快速切换，每个分页可以显示不同的内容。图 5-65 是 Android 系统内置的 Tab 标签页，单击“呼出/接听键”后出现，用于电话呼出和查看拨号记录、联系人。

1）Tab 标签页的使用

(1) 首先要设计所有分页的界面布局。

(2) 在分页设计完成后，使用代码建立 Tab 标签页，并给每个分页添加标识和标题。

(3) 最后确定每个分页所显示的界面布局。

(4) 每个分页建立一个 XML 文件，用以编辑和保存分页的界面布局，使用的方法与设计普通用户界面没有什么区别。

2）TabDemo 程序

建立一个名为 TabDemo 的程序，包含 3 个 XML 文件，分别为 tab1.xml、tab2.xml 和 tab3.xml，这 3 个文件分别使用线性布局、相对布局和绝对布局示例中的 main.xml 的代码，并将布局的 ID 分别定义为 layout01、layout02 和 layout03，如图 5-66 所示。



图 5-65 Tab 标签页



图 5-66 TabDemo 示意图

(1) tab1.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id = "@+id/layout01"
……
……
</LinearLayout>
```

(2) tab2.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout android:id="@+id/layout02"
  ……
  ……
</AbsoluteLayout>
```

(3) tab3.xml 文件代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:id="@+id/layout03"
  ……
  ……
</RelativeLayout>
```

(4) 在 TabDemo.java 文件中输入下面的代码，创建 Tab 标签页，并建立子页与界面布局直接的关联关系。

```
package edu.hrbeu.TabDemo;

import android.app.TabActivity;
import android.os.Bundle;
import android.widget.TabHost;
import android.view.LayoutInflater;

public class TabDemo extends TabActivity {
@Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      TabHost tabHost = getTabHost();
      LayoutInflater.from(this).inflate(R.layout.tab1,
            tabHost.getTabContentView(),true);
      LayoutInflater.from(this).inflate(R.layout.tab2,
            tabHost.getTabContentView(),true);
      LayoutInflater.from(this).inflate(R.layout.tab3,
            tabHost.getTabContentView(),true);
      tabHost.addTab(tabHost.newTabSpec("TAB1")
        .setIndicator("线性布局").setContent(R.id.layout01));
      tabHost.addTab(tabHost.newTabSpec("TAB2")
        .setIndicator("绝对布局").setContent(R.id.layout02));
```

```
20.     tabHost.addTab(tabHost.newTabSpec("TAB3")
21.       .setIndicator("相对布局").setContent(R.id.layout03));
22.           }
23.           }
```

第 8 行代码的声明 TabDemo 类继承于 TabActivity，与以往继承 Activity 不同，TabActivity 支持内嵌多个 Activity 或 View；

第 12 行代码通过 getTabHost()函数获得了 Tab 标签页的容器，用以承载可以单击的 Tab 标签和分页的界面布局；

第 13 行代码通过 LayoutInflater 将 tab1.xml 文件中的布局转换为 Tab 标签页可以使用的 View 对象；

第 16 行代码使用 addTab()函数添加了第 1 个分页，tabHost.newTabSpec("TAB1")表明在第 12 行代码中建立的 tabHost 上添加一个标识为 TAB1 的 Tab 分页；

第 17 行代码使用 setIndicator()函数设定分页显示的标题，使用 setContent()函数设定分页所关联的界面布局。

(5) TabDemo 示例的运行结果如图 5-67 所示。

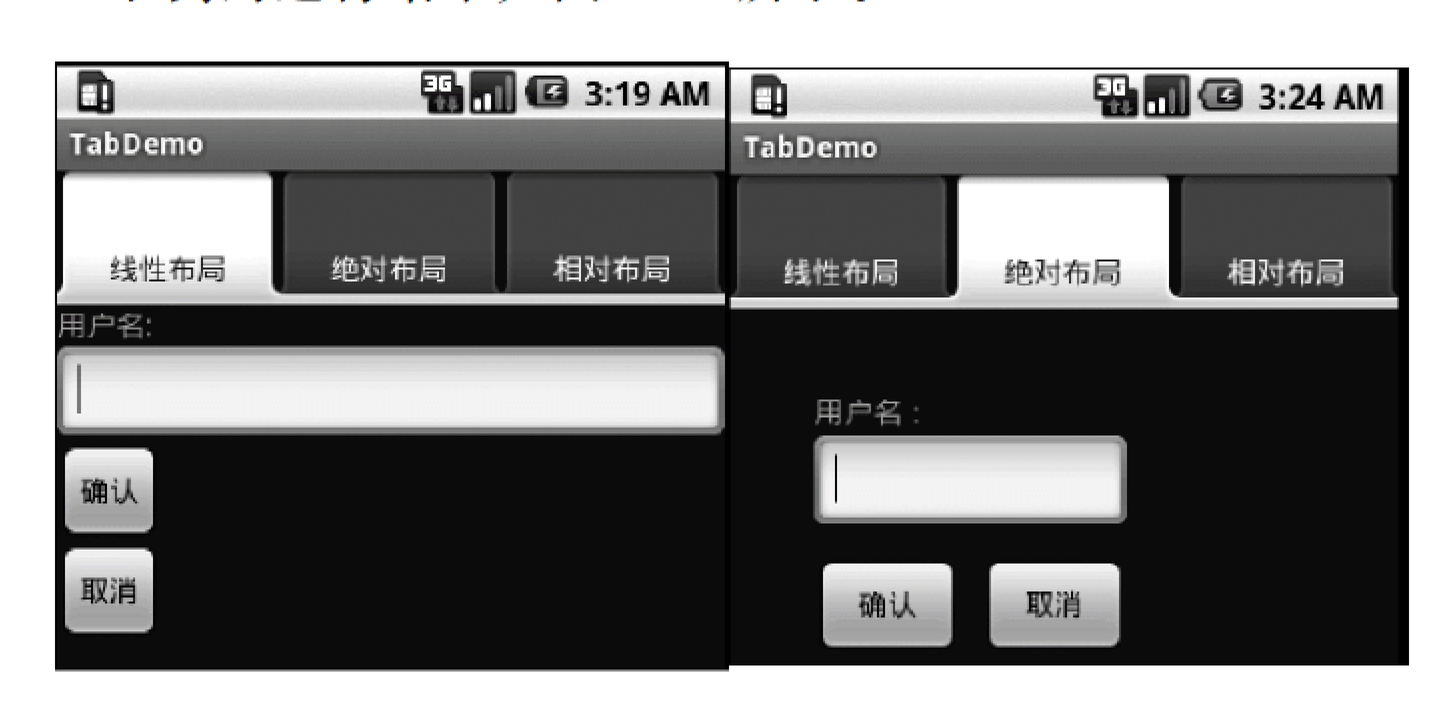


图 5-67　TabDemo 示例运行结果

(6) 在使用 Tab 标签页时，可以将不同分页的界面布局保存在不同的 XML 文件中，也可以将所有分页的布局保存在同一个 XML 文件中。第一种方法有利于在 Eclipse 开发环境中进行可视化设计，并且不同分页的界面布局在不同的文件中更加易于管理；第二种方法则可以产生较少的 XML 文件，同时编码时的代码也会更加简洁。

5.4.3　界面布局

界面布局(Layout)是用户界面结构的描述，定义了界面中所有的元素、结构和相互关系。声明 Android 程序的界面布局有两种方法：使用 XML 文件描述界面布局；在程序运行时动态添加或修改界面布局。用户既可以独立使用任何一种声明界面布局的方式，也可以同时使用两种方式。

使用 XML 文件声明界面布局的特点如下：

(1) 将程序的表现层和控制层分离。

(2) 在后期修改用户界面时无须更改程序的源代码。

(3) 用户还能够通过可视化工具直接看到所设计的用户界面，有利于加快界面设计的

过程，并且为界面设计与开发带来极大的便利性。

1. 线性布局

线性布局(LinearLayout)是一种重要的界面布局，也是经常使用到的一种界面布局。在线性布局中，所有的子元素都按照垂直或水平的顺序在界面上排列。如果垂直排列，则每行仅包含一个界面元素；如果水平排列，则每列仅包含一个界面元素。

1) 创建 Android 项目

创建一个 Android 项目，名称为 LinearLayout，包名称为 edu.hrbeu.LinearLayout，Activity 名称为 LinearLayout。

2) 建立 main_vertical.xml 文件

为了能够完整体验创建线性布局的过程，首先删除 Eclipse 自动建立的/res/layout/main.xml 文件，然后建立用于显示垂直排列线性布局的 XML 文件。操作步骤如下：

(1) 右击/res/layout 文件夹，在弹出的快捷菜单中选择 New → File 命令打开新文件建立向导。

(2) 设置文件名为 main_vertical.xml，保存位置为 LinearLayout/res/layout。

3) 界面布局的可视化编辑器

双击新建立的/res/layout/main_vertical.xml 文件，Eclipse 将打开界面布局的可视化编辑器，如图 5-68 所示。

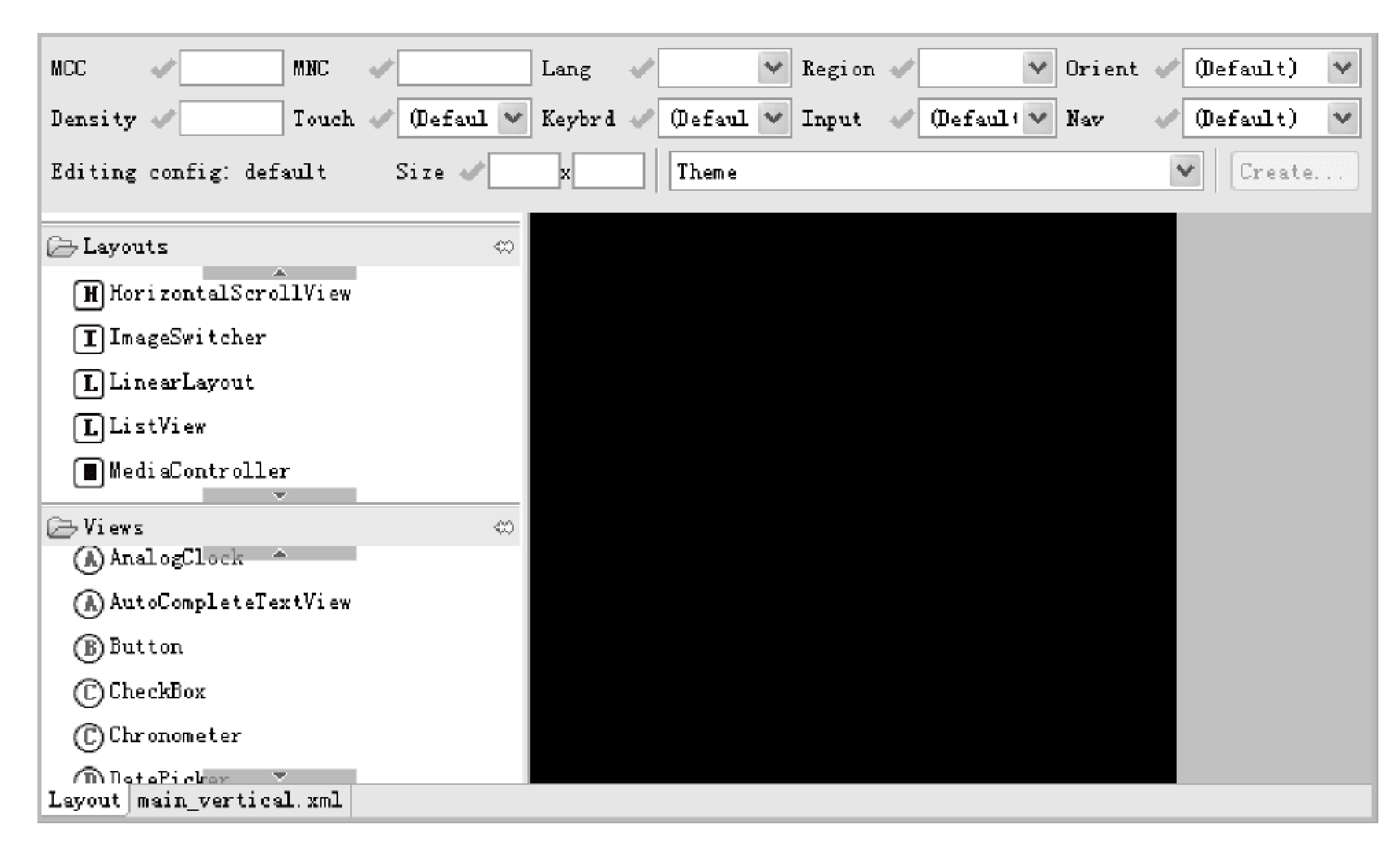


图 5-68 界面布局的可视化编辑器

可视化编辑器顶部是资源配置清单，可以根据手机的配置不同选择不同的资源，主要用来实现应用软件的本地化。

下部左侧是界面布局和界面控件，用户可以将需要的布局和控件拖曳到右面的可视化界面中，并修改布局和控件的属性。

右侧是可视化的用户界面，能够实时地呈现用户界面，但却无法正确显示中文。左下角的 Layout 和 main_vertical.xml 能够在可视化编辑器和 XML 文件编辑器之间切换。

4）线性布局的属性编辑器

在 Eclipse 右边的 Outline 中，双击 LinearLayout，打开线性布局的属性编辑器。

线性布局的排列方法主要由 Orientation 属性进行控制，vertical 表示垂直排列，horizontal 表示水平排列。选择 Orientation 的值为 vertical，表示该线性布局为垂直排列。

默认情况下，Layout height 的值为 wrap_content，表示线性布局高度等于所有子控件的高度总和，也就是线性布局的高度会刚好将所有子控件包含其中。

将 Layout width 属性的值改为 fill_parent，表示线性布局宽度等于父控件的宽度，就是将线性布局在横向上占据父控件的所有空间。

5）main_vertical. xml 文件代码

打开 XML 文件编辑器，main_vertical. xml 文件的代码如下：

```
1. <?xml version = "1.0" encoding = "utf - 8"?>
2. < LinearLayout
3.   xmlns:android = "http://schemas.android.com/apk/res/android"
4.   android:layout_width = "fill_parent"
5.   android:layout_height = "wrap_content"
6.   android:orientation = "vertical">
7.   </LinearLayout >
```

第 2 行代码是声明 XML 文件的根元素为线性布局；第 4、5、6 行代码是在属性编辑器中修改过的宽度、高度和排列方式的属性（用户在可视化编辑器和属性编辑器中的任何修改，都会同步地反映在 XML 文件中，反之亦然）。

6）界面控件及属性修改

将 4 个界面控件 TextView、EditText、Button、Button 先后拖曳到可视化编辑器中，所有控件都自动获取控件名称，并把该名称显示在控件上，例如 TextView01、EditText01、Button01 和 Button02。

修改界面控件的属性如表 5-1 所示。

表 5-1　修改界面控件的属性

编　　号	类　　型	属　　性	值
1	TextView	Id	@+id/label
		Text	用户名：
2	EditText	Id	@+id/entry
		Layout width	fill_parent
		Text	[null]
3	Button	Id	@+id/ok
		Text	确认
4	Button	Id	@+id/cancel
		Text	取消

所有界面控件都有一个共同的属性 Id。Id 是一个字符串，编译时被转换为整数，可以用来在代码中引用界面元素。一般仅在代码中需要动态修改的界面元素才为其设置 Id 属性，反之则不需要设置 Id 属性。

7）正常显示界面控件上的中文字符

从可视化编辑器中发现，界面控件上的中文字符都显示为"□"，因为可视化编辑器还不能很好地支持中文字符。

打开 XML 文件编辑器查看 main_vertical.xml 文件代码，发现在属性编辑器内填入的文字已经正常写入到 XML 文件中，例如第 11、20、25 行代码。

```
1. <?xml version = "1.0" encoding = "utf - 8"?>
2. < LinearLayout
3.   xmlns:android = "http://schemas.android.com/apk/res/android"
4.   android:layout_width = "fill_parent"
5.   android:layout_height = "wrap_content"
6.   android:orientation = "vertical">
7.
8. < TextView android:id = "@ + id/label"
9.      android:layout_width = "wrap_content"
10.      android:layout_height = "wrap_content"
11.               android:text = "用户名:" >
12.   </TextView >
13.      < EditText android:id = "@ + id/entry"
14.      android:layout_height = "wrap_content"
15.      android:layout_width = "fill_parent">
16. </EditText >
17. < Button android:id = "@ + id/ok"
18.      android:layout_width = "wrap_content"
19.      android:layout_height = "wrap_content"
20.      android:text = "确认">
21. </Button >
22. < Button android:id = "@ + id/cancel"
23.      android:layout_width = "wrap_content"
24.      android:layout_height = "wrap_content"
25.      android:text = "取消" >
26. </Button >
27. </LinearLayout >
```

将 LinearLayout.java 文件中的 setContentView(R.layout.main)更改为 setContentView(R.layout.main_vertical)。运行后的结果如图 5-69 所示。

8）横向线性布局

建立横向线性布局与建立纵向线性布局相似，只需注意以下几点：

（1）建立 main_ horizontal.xml 文件。

（2）线性布局的 Orientation 属性的值设置为 horizontal。

（3）将 EditText 的 Layout width 属性的值设置为 wrap_content。



图 5-69　运行结果

(4) 将 LinearLayout. java 文件中的 setContentView(R. layout. main_vertical) 修改为 setContentView(R. layout. main_ horizontal)。

2. 框架布局

框架布局(FrameLayout)是最简单的界面布局,用来存放一个元素的空白空间,且子元素的位置是不能够指定的,只能够放置在空白空间的左上角。如果有多个子元素,后放置的子元素将遮挡先放置的子元素。

使用 Android SDK 中提供的层级观察器(Hierarchy Viewer)可以进一步分析界面布局,它能够对用户界面进行分析和调试,并以图形化的方式展示树形结构的界面布局。Android SDK 还提供了一个精确的像素级观察器(Pixel Perfect View),以栅格的方式详细观察放大后的界面布局。

在层级观察器中获得示例界面布局的树形结构图如图 5-70 所示。

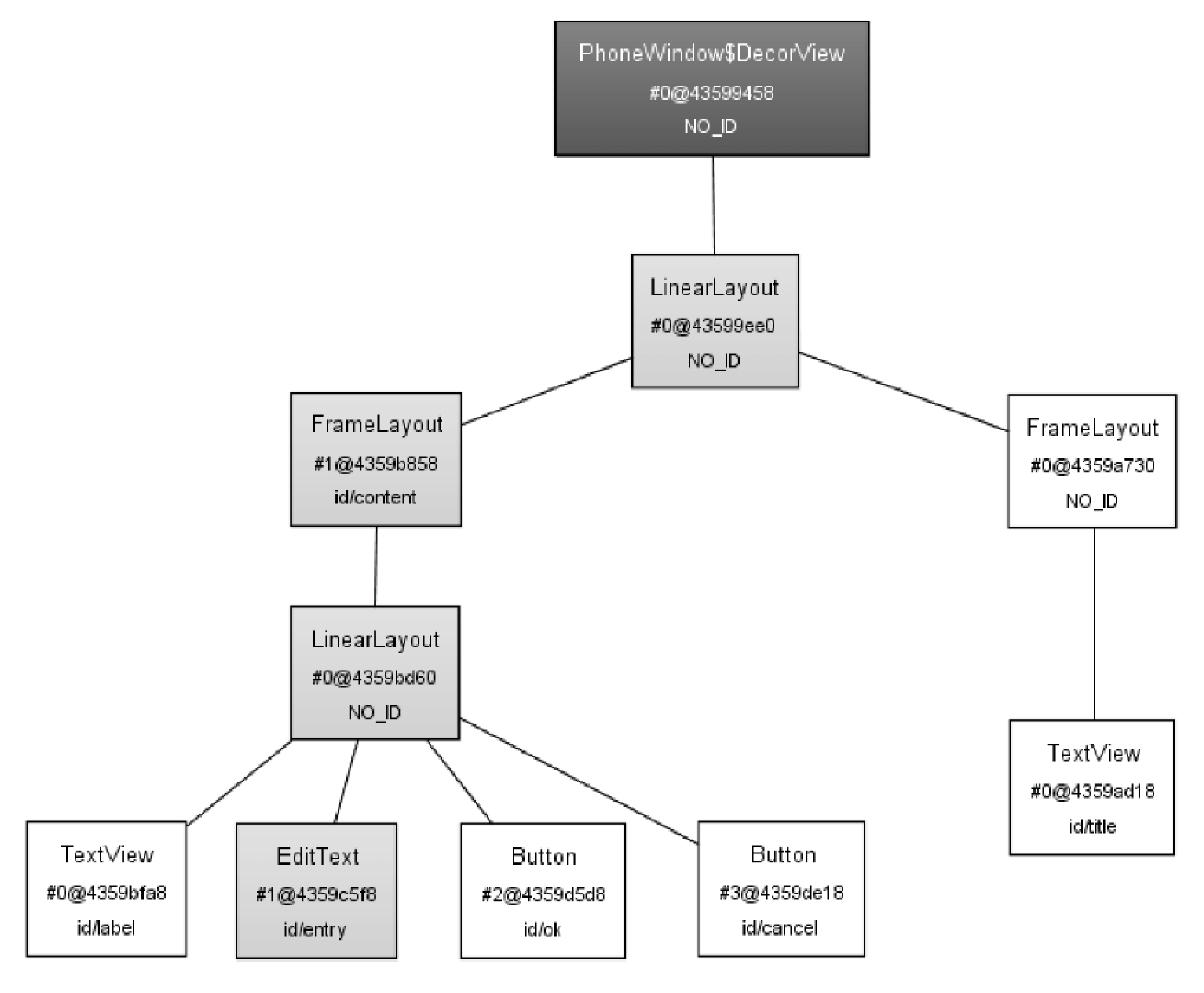


图 5-70　界面布局的树形结构图

结合界面布局的树形结构图和示意图,分析不同界面布局和界面控件的区域边界。用户界面的根节点(＃0@43599ee0)是线性布局,其边界是整个界面,也就是示意图最外层的实心线。

根节点右侧的子节点(＃0@4359a730)是框架布局,仅有一个节点元素(＃0@4359ad18),这个子元素是 TextView 控件,用来显示 Android 应用程序名称,其边界是示意图中的区域 1。因此框架布局元素＃0@4359a730 的边界与区域 1 的高度相同,宽度充满整个根节点的

区域。这两个界面元素是系统自动生成的，一般情况下用户不能修改和编辑。

根节点左侧的子节点（＃1@4359b858）也是框架布局，边界是区域 2 到区域 7 的全部空间。

子节点（＃1@4359b858）下仅有一个子节点（＃0@4359bd60）元素是线性布局，因为线性布局的 Layout width 属性设置为 fill_parent，Layout height 属性设置为 wrap_content，因此该线性布局的宽度就是其父节点＃1@4359b858 的宽度，高度等于所有子节点元素的高度之和。

线性布局＃0@4359bd60 的 4 个子节点元素＃0@4359bfa8、＃1@4359c5f8、＃2@4359d5d8 和＃3@4359de18 的边界，分别是界面布局示意图中的区域 2、区域 3、区域 4 和区域 5。

3. 表格布局

表格布局（TableLayout）也是一种常用的界面布局，它将屏幕划分网格，通过指定行和列可以将界面元素添加到网格中，网格的边界对用户是不可见的。

表格布局还支持嵌套，可以将另一个表格布局放置在前一个表格布局的网格中，也可以在表格布局中添加其他界面布局，例如线性布局、相对布局等。

表格布局示意图如图 5-71 所示，表格布局效果图如图 5-72 所示。

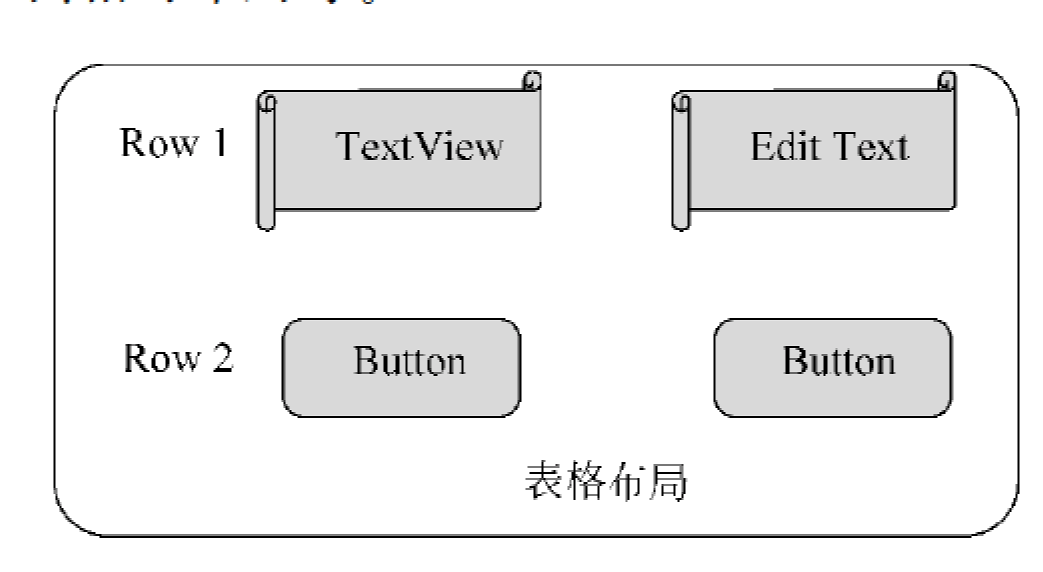


图 5-71　表格布局示意图

建立如图 5-72 所示表格布局（TableLayout）的过程如下：

（1）向界面中添加一个线性布局，无须修改布局的属性值。其中，Id 属性为 TableLayout01，Layout width 和 Layout height 属性都为 wrap_content。

（2）向 TableLayout01 中添加两个 TableRow。TableRow 代表一个单独的行，每行被划分为几个小的单元，单元中可以添加一个界面控件。其中，Id 属性分别为 TableRow01 和 TableRow02，Layout width 和 Layout height 属性都为 wrap_content。

（3）通过 Outline 向 TableRow01 中添加 TextView 和 EditText，如图 5-73 所示。



图 5-72　表格布局效果图



图 5-73　添加 TableRow

（4）同样的方法，通过 Outline 向 TableRow02 中添加两个 Button。

（5）参考表 5-2 设置 TableRow 中 4 个界面控件的属性值。

表 5-2　TableRow 中 4 个界面控件的属性值

编　号	类　型	属　性	值
1	TextView	Id	@+id/label
		Text	用户名：
		Gravity	right
		Padding	3dip
		Layout width	160dip
2	EditText	Id	@+id/entry
		Text	[null]
		Padding	3dip
		Layout width	160dip
3	Button	Id	@+id/ok
		Text	确认
		Padding	3dip
4	Button	Id	@+id/cancel
		Text	取消
		Padding	3dip

（6）main. xml 文件的完整代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>

< TableLayout android:id = "@ + id/TableLayout01"
  android:layout_width = "fill_parent"
  android:layout_height = "fill_parent"
  xmlns:android = "http://schemas.android.com/apk/res/android">
  < TableRow android:id = "@ + id/TableRow01"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content">
     < TextView android:id = "@ + id/label"
        android:layout_height = "wrap_content"
        android:layout_width = "160dip"
        android:gravity = "right"
        android:text = "用户名: "
        android:padding = "3dip" >
     </TextView >
                              < EditText android:id = "@ + id/entry"
        android:layout_height = "wrap_content"
        android:layout_width = "160dip"
        android:padding = "3dip" >
     </EditText >
  </TableRow >
```

```
23.   <TableRow android:id = "@ + id/TableRow02"
24.     android:layout_width = "wrap_content"
25.     android:layout_height = "wrap_content">
26.     <Button android:id = "@ + id/ok"
27.       android:layout_height = "wrap_content"
28.       android:padding = "3dip"
29.       android:text = "确认">
30.     </Button>
31.                                <Button android:id = "@ + id/Button02"
32.       android:layout_width = "wrap_content"
33.       android:layout_height = "wrap_content"
34.       android:padding = "3dip"
35.       android:text = "取消">
36.     </Button>
37.   </TableRow>
38.   </TableLayout>
```

第 3 行代码使用了＜TableLayout＞标签声明表格布局；第 7 行和第 23 行代码声明了两个 TableRow 元素；第 12 行设定宽度属性 android:layout_width：160dip；第 13 行设定属性 android：gravity，指定文字为右对齐；第 15 行使用属性 android：padding，声明 TextView 元素与其他元素的间隔距离为 3dip。

4. 相对布局

相对布局(RelativeLayout)是一种非常灵活的布局方式，能够通过指定界面元素与其他元素的相对位置关系确定界面中所有元素的布局位置。其特点是能够最大程度地保证在各种屏幕类型的手机上正确显示界面布局。

相对布局(RelativeLayout)示例说明如图 5-74 所示。其创建过程如下：

(1) 添加 TextView 控件(“用户名”)，相对布局会将 TextView 控件放置在屏幕的最上方。

(2) 然后添加 EditText 控件(输入框)，并声明该控件的位置在 TextView 控件的下方，相对布局会根据 TextView 的位置确定 EditText 控件的位置。

(3) 之后添加第一个 Button 控件(“取消”按钮)，声明在 EditText 控件的下方，且在父控件的最右边。



图 5-74 相对布局示例说明

(4) 最后，添加第二个 Button 控件(“确认”按钮)，声明该控件在第一个 Button 控件的左方，且与第一个 Button 控件处于相同的水平位置。

(5) 相对布局的 main. xml 文件的完整代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<RelativeLayout android:id = "@ + id/RelativeLayout01"
```

```
3.  android:layout_width = "fill_parent"
4.  android:layout_height = "fill_parent"
5.  xmlns:android = "http://schemas.android.com/apk/res/android">
6.  <TextView android:id = "@ + id/label"
7.    android:layout_height = "wrap_content"
8.    android:layout_width = "fill_parent"
9.    android:text = "用户名: ">
10.  </TextView>
11.  <EditText android:id = "@ + id/entry"
12.    android:layout_height = "wrap_content"
13.    android:layout_width = "fill_parent"
14.    android:layout_below = "@id/label">
15.  </EditText>
16.            <Button android:id = "@ + id/cancel"
17.                        android:layout_height = "wrap_content"
18.    android:layout_width = "wrap_content"
19.    android:layout_alignParentRight = "true"
20.    android:layout_marginLeft = "10dip"
21.    android:layout_below = "@id/entry"
22.    android:text = "取消" >
23.  </Button>
24.    <Button android:id = "@ + id/ok"
25.    android:layout_height = "wrap_content"
26.    android:layout_width = "wrap_content"
27.    android:layout_toLeftOf = "@id/cancel"
28.    android:layout_alignTop = "@id/cancel"
29.    android:text = "确认">、
30.  </Button>
31. </RelativeLayout>
```

第 3 行使用了<RelativeLayout>标签声明一个相对布局;

第 15 行使用位置属性 android:layout_below,确定 EditText 控件在 ID 为 label 的元素下方;

第 19 行使用属性 android:layout_alignParentRight,声明该元素在其父元素的右边边界对齐;

第 20 行设定属性 android:layout_marginLeft,左移 10dip;

第 21 行声明该元素在 ID 为 entry 的元素下方;

第 27 行声明使用属性 android:layout_toLeftOf,声明该元素在 ID 为 cancel 的元素的左边;

第 28 行使用属性 android:layout_alignTop,声明该元素与 ID 为 cancel 的元素在相同的水平位置。

5. 绝对布局

绝对布局(AbsoluteLayout)能通过指定界面元素的坐标位置来确定用户界面的整体布局。

绝对布局是一种不推荐使用的界面布局,因为通过 X 轴和 Y 轴确定界面元素位置后,

Android 系统不能够根据不同屏幕对界面元素的位置进行调整，这就降低了界面布局对不同类型和尺寸屏幕的适应能力。

绝对布局（AbsoluteLayout）示意图如图 5-75 所示，每一个界面控件都必须指定坐标（X，Y），例如“确认”按钮的坐标是（40，120），“取消”按钮的坐标是（120，120）。坐标原点（0，0）在屏幕的左上角。

绝对布局示例的 main.xml 文件的完整代码如下：



图 5-75　绝对布局示意图

```
<?xml version = "1.0" encoding = "utf - 8"?>

<AbsoluteLayout android:id = "@ + id/AbsoluteLayout01"
  android:layout_width = "fill_parent"
  android:layout_height = "fill_parent"
  xmlns:android = "http://schemas.android.com/apk/res/android">
  <TextView android:id = "@ + id/label"
    android:layout_x = "40dip"
    android:layout_y = "40dip"
     android:layout_height = "wrap_content"
     android:layout_width = "wrap_content"
     android:text = "用户名: ">
   </TextView>
   <EditText android:id = "@ + id/entry"
     android:layout_x = "40dip"
    android:layout_y = "60dip"
                      android:layout_height = "wrap_content"
                      android:layout_width = "150dip">
                </EditText>
                <Button android:id = "@ + id/ok"
                      android:layout_width = "70dip"
                      android:layout_height = "wrap_content"
                      android:layout_x = "40dip"
                      android:layout_y = "120dip"
                      android:text = "确认">
                </Button>
        <Button android:id = "@ + id/cancel"
                      android:layout_width = "70dip"
                      android:layout_height = "wrap_content"
                      android:layout_x = "120dip"
                      android:layout_y = "120dip"
                      android:text = "取消">
                 </Button>
    </AbsoluteLayout>
```

5.4.4 菜单

菜单是应用程序中非常重要的组成部分，能够在不占用界面空间的前提下为应用程序提供统一的功能和设置界面，并为程序开发人员提供易于使用的编程接口。

Android 系统支持 3 种菜单：选项菜单（Option Menu）、子菜单（Submenu）和快捷菜单（Context Menu）。

1. 选项菜单

选项菜单是一种经常被使用的 Android 系统菜单，通过“菜单键”（MENU key）打开。

1）选项菜单分类

（1）图标菜单（Icon Menu）。图标菜单是能够同时显示文字和图标的菜单，最多支持 6 个子项；图标菜单不支持单选框和复选框。

（2）扩展菜单（Expanded Menu）。扩展菜单在图标菜单子项多余 6 个时才会出现，通过单击图标菜单最后的子项 More 才能打开。扩展菜单是垂直的列表型菜单；不能够显示图标；支持单选框和复选框。

重载 Activity 的 onCreateOptionMenu()函数才能够在 Android 应用程序中使用选项菜单。初次使用选项菜单时，会调用 onCreateOptionMenu()函数，用来初始化菜单子项的相关内容，设置菜单子项自身子项的 ID 和组 ID、菜单子项显示的文字和图片等。

2）选项菜单代码示例

```
1.  final static int MENU_DOWNLOAD = Menu.FIRST;
2.  final static int MENU_UPLOAD = Menu.FIRST + 1;
3.    @Override
4.    public boolean onCreateOptionsMenu(Menu menu){
5.        menu.add(0,MENU_DOWNLOAD,0,"下载设置");
6.        menu.add(0,MENU_UPLOAD,1,"上传设置");
7.        return true;
    }
```

第 1 行和第 2 行代码将菜单子项 ID 定义成静态常量，并使用静态常量 Menu. FIRST（整数类型，值为 1）定义第一个菜单子项，以后的菜单子项仅需在 Menu. FIRST 增加相应的数值即可。

第 4 行代码 Menu 对象作为一个参数被传递到函数内部，因此在 onCreateOptionsMenu()函数中，用户可以使用 Menu 对象的 add()函数添加菜单子项。

第 7 行代码是 onCreateOptionsMenu()函数返回值，函数的返回值类型为布尔型。返回 true 将显示在函数中设置的菜单，否则不能够显示菜单。

3）add()函数的语法

```
MenuItem android.view.Menu.add(int groupId, int itemId, int order, CharSequence title).
```

（1）第一个参数 groupId 是组 ID，用以批量地对菜单子项进行处理和排序。

（2）第二个参数 itemId 是子项 ID，是每一个菜单子项的唯一标识，通过子项 ID 使应用程序能够定位到用户所选择的菜单子项。

(3) 第三个参数 order 是定义菜单子项在选项菜单中的排列顺序。

(4) 第四个参数 title 是菜单子项所显示的标题。

4) 添加菜单子项的图标和快捷键

使用 setIcon()函数和 setShortcut()函数。MENU_DOWNLOAD 菜单设置图标和快捷键的代码如下：

```
1.      menu.add(0,MENU_DOWNLOAD,0,"下载设置")
2.          .setIcon(R.drawable.download);
3. .setShortcut(','d');
```

第 2 行代码中使用了新的图像资源，用户将需要使用的图像文件复制到/res/drawable 目录下。

第 3 行中的 setShortcut()函数的第一个参数是为数字键盘设定的快捷键；第二个参数是为全键盘设定的快捷键，且不区分字母的大小写。

5) OnPrepareOptionsMenu()函数

重载 onPrepareOptionsMenu()函数能够动态地添加、删除菜单子项，或修改菜单的标题、图标和可见性等内容。

onPrepareOptionsMenu()函数的返回值的含义与 onCreateOptionsMenu()函数的相同，返回 true 则显示菜单，返回 false 则不显示菜单。

下面的代码是在用户每次打开选项菜单时，在菜单子项中显示用户打开该子项的次数。

```
1. static int MenuUploadCounter = 0;
2. @Override
3. public boolean onPrepareOptionsMenu(Menu menu){
4.      MenuItem uploadItem = menu.findItem(MENU_UPLOAD);
5.      uploadItem.setTitle("上传设置:" + String.valueOf(MenuUploadCounter));
6.      return true;
7.      }
```

第 1 行代码设置一个菜单子项的计数器，用来统计用户打开“上传设置”子项的次数；第 4 行代码是通过将菜单子项的 ID 传递给 menu.findItem()函数获取菜单子项的对象；第 5 行代码是通过 MenuItem 的 setTitle()函数修改菜单标题。

6) onOptionsItemSelected ()函数

onOptionsItemSelected ()函数能够处理菜单选择事件，且该函数在每次单击菜单子项时都会被调用。下面的代码说明了如何通过菜单子项的子项 ID 执行不同的操作。

```
@Override
public boolean onOptionsItemSelected(MenuItem item){
switch(item.getItemId()){
    case MENU_DOWNLOAD:
        MenuDownlaodCounter++;
        return true;
```

```
7.     case MENU_UPLOAD:
8.         MenuUploadCounter++;
9.         return true;
10.   }
11.   return false;
12.   }
```

onOptionsItemSelected()函数的返回值表示是否对菜单的选择事件进行处理,如果已经处理过则返回 true,否则返回 false;第 2 行的 MenuItem.getItemId()函数可以获取到被选择菜单子项的 ID。完整代码请参考 OptionsMenu 程序。

程序运行后,通过单击“菜单键”可以调出程序设计的两个菜单子项。

2. 子菜单

子菜单是能够显示更加详细信息的菜单子项。菜单子项使用了浮动窗体的显示形式,能够更好地适应小屏幕的显示方式,如图 5-76 所示。



图 5-76　菜单子项

Android 系统的子菜单使用非常灵活,可以在选项菜单或快捷菜单中使用子菜单,有利于将相同或相似的菜单子项组织在一起,便于显示和分类。子菜单不支持嵌套;子菜单的添加是使用 addSubMenu()函数实现的。子菜单的程序代码示例如下:

```
1. SubMenu uploadMenu = (SubMenu) menu.addSubMenu(0,MENU_UPLOAD,1,"上传设置")
       .setIcon(R.drawable.upload);
2. uploadMenu.setHeaderIcon(R.drawable.upload);
3. uploadMenu.setHeaderTitle("上传参数设置");
4. uploadMenu.add(0,SUB_MENU_UPLOAD_A,0,"上传参数 A");
5. uploadMenu.add(0,SUB_MENU_UPLOAD_B,0,"上传参数 B");
```

第 1 行代码在 onCreateOptionsMenu()函数传递的 menu 对象上调用 addSubMenu()函数,在选项菜单中添加一个菜单子项,用户单击后可以打开子菜单;addSubMenu()函数与选项菜单中使用过的 add()函数支持相同的参数,同样可以指定菜单子项的 ID、组 ID 和标题等参数,并且能够通过 setIcon()函数菜单所显示的图标。

第 2 行代码使用 setHeaderIcon ()函数,定义子菜单的图标。

第 3 行定义子菜单的标题。若不规定子菜单的标题,子菜单将显示父菜单子项标题,即第 1 行代码中的“上传设置”。

第 4 行和第 5 行在子菜单中添加了两个菜单子项,菜单子项的更新函数和选择事件处理函数仍然使用 onPrepareOptionsMenu()函数和 onOptionsItemSelected()函数。

以上小节的代码为基础,将“上传设置”改为子菜单,并在子菜单中添加“上传参数 A”和

"上传参数 B"两个菜单子项。完整代码请参考 MySubMenu 程序,运行结果如图 5-77 所示。



图 5-77 子菜单程序 MySubMenu 运行结果

3. 快捷菜单

快捷菜单同样采用了动态窗体的显示方式,与子菜单的实现方式相同,但两种菜单的启动方式却截然不同。

1) 快捷菜单的启动方式和使用方法

(1) 启动方式:快捷菜单类似于普通桌面程序中的右键菜单,当用户单击界面元素超过 2 秒后,将启动注册到该界面元素的快捷菜单。

(2) 使用方法:与使用选项菜单的方法非常相似,需要重载 onCreateContextMenu()函数和 onContextItemSelected()函数。

2) onCreateContextMenu()函数

onCreateContextMenu()函数主要用来添加快捷菜单所显示的标题、图标和菜单子项等内容。选项菜单中的 onCreateOptionsMenu()函数仅在选项菜单第一次启动时被调用一次,而快捷菜单的 onCreateContextMenu()函数每次启动时都会被调用一次。示例代码如下:

```
1. final static int CONTEXT_MENU_1 = Menu.FIRST;
2. final static int CONTEXT_MENU_2 = Menu.FIRST + 1;
3. final static int CONTEXT_MENU_3 = Menu.FIRST + 2;
4. @Override
5. public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo){
6.   menu.setHeaderTitle("快捷菜单标题");
7.   menu.add(0, CONTEXT_MENU_1, 0,"菜单子项 1");
8.   menu.add(0, CONTEXT_MENU_2, 1,"菜单子项 2");
9.   menu.add(0, CONTEXT_MENU_3, 2,"菜单子项 3");
10. }
```

ContextMenu 类支持 add()函数(代码第 7 行)和 addSubMenu()函数,可以在快捷菜单中添加菜单子项和子菜单。

第 5 行代码的 onCreateContextMenu()函数中的参数:第一个参数 menu 是需要显示的快捷菜单;第二个参数 v 是用户选择的界面元素;第三个参数 menuInfo 是所选择界面元素的额外信息。

3) onContextItemSelected()函数

菜单选择事件的处理需要重载 onContextItemSelected()函数,该函数在用户选择快捷菜单中的菜单子项后被调用,与 onOptionsItemSelected ()函数的使用方法基本相同。示例代码如下:

```
1. @Override
2. public boolean onContextItemSelected(MenuItem item){
3.   switch(item.getItemId()){
4.     case CONTEXT_MENU_1:
5.       LabelView.setText("菜单子项 1");
6.       return true;
7.     case CONTEXT_MENU_2:
8.       LabelView.setText("菜单子项 2");
9.       return true;
10.     case CONTEXT_MENU_3:
11.       LabelView.setText("菜单子项 3");
12.   return true;
13.   }
14.   return false;
15.   }
```

4) 启动快捷菜单

使用 registerForContextMenu()函数将快捷菜单注册到界面控件上(下方代码第 7 行),用户在长时间单击该界面控件时,便会启动快捷菜单。

为了能够在界面上直接显示用户所选择快捷菜单的菜单子项,在代码中引用了界面元素 TextView(下方代码第 6 行),通过更改 TextView 的显示内容(上方代码第 5、8 和 11 行)来显示用户所选择的菜单子项。

```
1. TextView LabelView = null;
2. @Override
3. public void onCreate(Bundle savedInstanceState) {
4.   super.onCreate(savedInstanceState);
5.   setContentView(R.layout.main);
6.   LabelView = (TextView)findViewById(R.id.label);
7.   registerForContextMenu(LabelView);
8.   }
```

下方代码是/src/layout/main.xml 文件的部分内容,第 1 行声明了 TextView 的 ID 为 label;在上方代码的第 6 行中,通过 R.id.label 将 ID 传递给 findViewById()函数,这样用户便能够引用该界面元素,并能够修改该界面元素的显示内容。

```
1. <TextView android:id="@+id/label"
2.     android:layout_width="fill_parent"
3.     android:layout_height="fill_parent"
4.     android:text="@string/hello"
5. />
```

需要注意的是,上方代码的第 2 行将 android:layout_width 设置为 fill_parent,这样 TextView 将填充满父节点的所有剩余屏幕空间,用户单击屏幕 TextView 下方任何位置都可以启动快捷菜单;如果将 android:layout_width 设置为 wrap_content,则用户必须准确单击 TextView 才能启动快捷菜单。



图 5-78　MyContextMenu 程序运行结果

完整代码请参考 MyContextMenu 程序，运行结果如图 5-78 所示。

5）使用 XML 文件定义菜单

在 Android 系统中，菜单不仅能够在代码中定义，而且可以像界面布局一样在 XML 文件中进行定义。

使用 XML 文件定义界面菜单，将代码与界面设计分类，有助于简化代码的复杂程度，并且更有利于界面的可视化。

下面将快捷菜单的示例程序 MyContextMenu 改用 XML 实现，新程序的工程名称为 MyXLMContextMenu。

首先需要创建保存菜单内容的 XML 文件，在/src 目录下建立子目录 menu，并在 menu 下建立 context_menu. xml 文件，代码如下：

```
1. <menu xmlns:android = "http://schemas.android.com/apk/res/android">
2.   <item android:id = "@ + id/contextMenu1"
3.     android:title = "菜单子项 1"/>
4.   <item android:id = "@ + id/contextMenu2"
5.     android:title = "菜单子项 2"/>
6.   <item android:id = "@ + id/contextMenu3"
7.     android:title = "菜单子项 3"/>
8.   </menu>
```

在描述菜单的 XML 文件中必须以<menu>标签（代码第 1 行）作为根节点，<item>标签（代码第 2 行）用来描述菜单中的子项，<item>标签可以通过嵌套实现子菜单的功能。

XML 菜单的显示结果如图 5-79 所示。

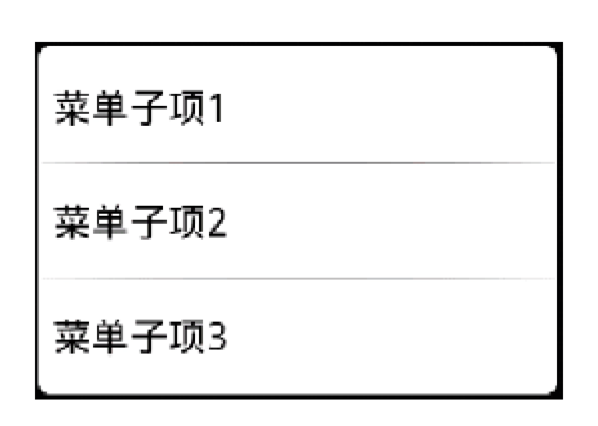


图 5-79　XML 菜单显示结果

在 XML 文件中定义菜单后，在 onCreateContextMenu() 函数中调用 inflater. inflate() 方法，将 XML 资源文件传递给菜单对象。示例代码如下：

```
1. @Override
2. public void onCreateContextMenu(ContextMenu menu,
3.     View v, ContextMenuInfo menuInfo){
4.   MenuInflater inflater = getMenuInflater();
5.   inflater.inflate(R.menu.context_menu, menu);
6.   }
```

第 4 行代码中的 getMenuInflater() 为当前的 Activity 返回 MenuInflater；第 5 行代码将 XML 资源文件 R. menu. context_menu 传递给 menu 这个快捷菜单对象。

5.4.5 界面事件

在 Android 系统中存在多种界面事件，例如单击事件、触摸事件、焦点事件和菜单事件等。在这些界面事件发生时，Android 界面框架调用界面控件的事件处理函数对事件进行处理。

1. 按键事件

在 MVC 模型中，控制器根据界面事件(UI Event)类型不同，将事件传递给界面控件不同的事件处理函数。按键事件(KeyEvent)将传递给 onKey()函数进行处理；触摸事件(TouchEvent)将传递给 onTouch()函数进行处理。

1) 事件监听器

Android 系统界面事件的传递和处理遵循一定的规则：

(1) 如果界面控件设置了事件监听器，则事件将先传递给事件监听器。

(2) 如果界面控件没有设置事件监听器，界面事件则会直接传递给界面控件的其他事件处理函数；即使界面控件设置了事件监听器，界面事件也可以再次传递给其他事件处理函数；是否继续传递事件给其他处理函数是由事件监听器处理函数的返回值决定的。

(3) 如果监听器处理函数的返回值为 true，则表示该事件已经完成处理过程，不需要其他处理函数参与处理过程，这样事件就不会再继续进行传递。

(4) 如果监听器处理函数的返回值为 false，则表示该事件没有完成处理过程，或需要其他处理函数捕获到该事件，事件会被传递给其他事件处理函数。

以 EditText 控件中的按键事件为例，说明 Android 系统界面事件传递和处理过程。假设 EditText 控件已经设置了按键事件监听器。当用户按下键盘上的某个按键时，控制器将产生 KeyEvent 按键事件，Android 系统会首先判断 EditText 控件是否设置了按键事件监听器；因为 EditText 控件已经设置按键事件监听器 OnKeyListener，所以按键事件先传递到监听器的事件处理函数 onKey()中；事件能否继续传递给 EditText 控件的其他事件处理函数，完全根据 onKey()函数的返回值来确定。

如果 onKey()函数返回 false，事件将继续传递，这样 EditText 控件就可以捕获到该事件，将按键的内容显示在 EditText 控件中。

如果 onKey()函数返回 true，将阻止按键事件的继续传递，这样 EditText 控件就不能够捕获到按键事件，也就不能够将按键内容显示在 EditText 控件中。

Android 界面框架支持对按键事件的监听，并能够将按键事件的详细信息传递给处理函数。为了处理控件的按键事件，先需要设置按键事件的监听器，并重载 onKey()函数。示例代码如下：

```
1. entryText.setOnKeyListener(new OnKeyListener(){
2.   @Override
3.   public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4.     //过程代码……
5.     return true/false;
6.     }
```

第 1 行代码是设置控件的按键事件监听器。

第 3 行代码的 onKey ()函数中的参数：第一个参数 view 表示产生按键事件的界面控件；第二个参数 keyCode 表示按键代码；第三个参数 keyEvent 则包含了事件的详细信息，如按键的重复次数、硬件编码和按键标志等。

第 5 行代码是 onKey ()函数的返回值；返回 true 则阻止事件传递；返回 false 则允许继续传递按键事件。

2）KeyEventDemo 用户界面

KeyEventDemo 程序是一个说明如何处理按键事件的示例，其用户界面最上方的 EditText 控件是输入字符的区域；中间的 CheckBox 控件用来控制 onKey()函数的返回值；最下方的 TextView 控件用来显示按键事件的详细信息，包括按键动作、按键代码、按键字符、UNICODE 编码、重复次数、功能键状态、硬件编码和按键标志，如图 5-80 所示。



图 5-80　KeyEventDemo 用户界面

3）KeyEventDemo 用户界面的 XML 文件代码

```
1. <EditText android:id="@+id/entry"
2.   android:layout_width="fill_parent"
3.   android:layout_height="wrap_content">
4. </EditText>
5. <CheckBox android:id="@+id/block"
6.   android:layout_width="wrap_content"
7.   android:layout_height="wrap_content"
8.   android:text="返回 true,阻止将按键事件传递给界面元素" >
9. </CheckBox>
10. <TextView android:id="@+id/label"
11.   android:layout_width="wrap_content"
12.   android:layout_height="wrap_content"
13.   android:text="按键事件信息" >
14. </TextView>
```

4）按键事件的处理

在 EditText 中，每当任何一个按键按下或抬起时都会引发按键事件。为了能够使 EditText 处理按键事件，需要使用 setOnKeyListener ()函数在代码中设置按键事件监听器，并在 onKey()函数添加按键事件的处理过程。代码如下：

```
1. entryText.setOnKeyListener(new OnKeyListener(){
2. @Override
3. public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4. int metaState = keyEvent.getMetaState();
5. int unicodeChar = keyEvent.getUnicodeChar();
6. String msg = "";
7. msg += "按键动作:" + String.valueOf(keyEvent.getAction()) + "\n";
8. msg += "按键代码:" + String.valueOf(keyCode) + "\n";
9. msg += "按键字符:" + (char)unicodeChar + "\n";
10. msg += "UNICODE:" + String.valueOf(unicodeChar) + "\n";
```

```
11. msg += "重复次数:" + String.valueOf(keyEvent.getRepeatCount()) + "\n";
12. msg += "功能键状态:" + String.valueOf(metaState) + "\n";
13. msg += "硬件编码:" + String.valueOf(keyEvent.getScanCode()) + "\n";
14. msg += "按键标志:" + String.valueOf(keyEvent.getFlags()) + "\n";
15. labelView.setText(msg);
16. if (checkBox.isChecked())
17.   return true;
18. else
19.   return false;
20. }
```

第 4 行代码用来获取功能键状态。功能键包括左 Alt 键、右 Alt 键和 Shift 键，当这 3 个功能键被按下时，功能键代码 metaState 值分别为 18、34 和 65；但没有功能键被按下时，功能键代码 metaState 值分别为 0。

第 5 行代码获取了按键的 UNICODE 值；在第 9 行中将 UNICODE 转换为字符，显示在 TextView 中。

第 7 行代码获取了按键动作，0 表示按下按键，1 表示抬起按键。

第 11 行代码获取按键的重复次数，但按键被长时间按下时则会产生这个属性值。

第 13 行代码获取了按键的硬件编码，不同硬件设备的按键硬件编码都不相同，因此该值一般用于调试。

第 14 行获取了按键事件的标识符。

2. 触摸事件

1）触摸事件监听器

Android 界面框架支持对触摸事件的监听，并能够将触摸事件的详细信息传递给处理函数，需要设置触摸事件的监听器，并重载 onTouch ()函数。示例代码如下：

```
1. touchView.setOnTouchListener(new View.OnTouchListener(){
2.   @Override
3.   public boolean onTouch(View v, MotionEvent event) {
4.     //过程代码……
5.     return true/false;
6.     }
```

第 1 行代码是设置控件的触摸事件监听器。

在第 3 行代码的 onTouch()函数中，第一个参数 View 表示产生触摸事件的界面控件；第二个参数 MontionEvent 表示触摸事件的详细信息，例如产生时间、坐标和触点压力等。

第 5 行是 onTouch()函数的返回值。

2）TouchEventDemo 用户界面

TouchEventDemo 是一个说明如何处理触摸事件的示例，其用户界面如图 5-81 所示。

上方浅色区域是可以接受触摸事件的区域，用户可以在 Android Emulator 中使用鼠标单击屏幕，用以模拟触摸手机屏幕；下方深色区域是显示区域，用来显示触摸事件的类型、相对坐标、绝对坐标、触点压力、触点尺寸和历史数据量等信息。右边白色区域文字是左图深色区域显示的文字。

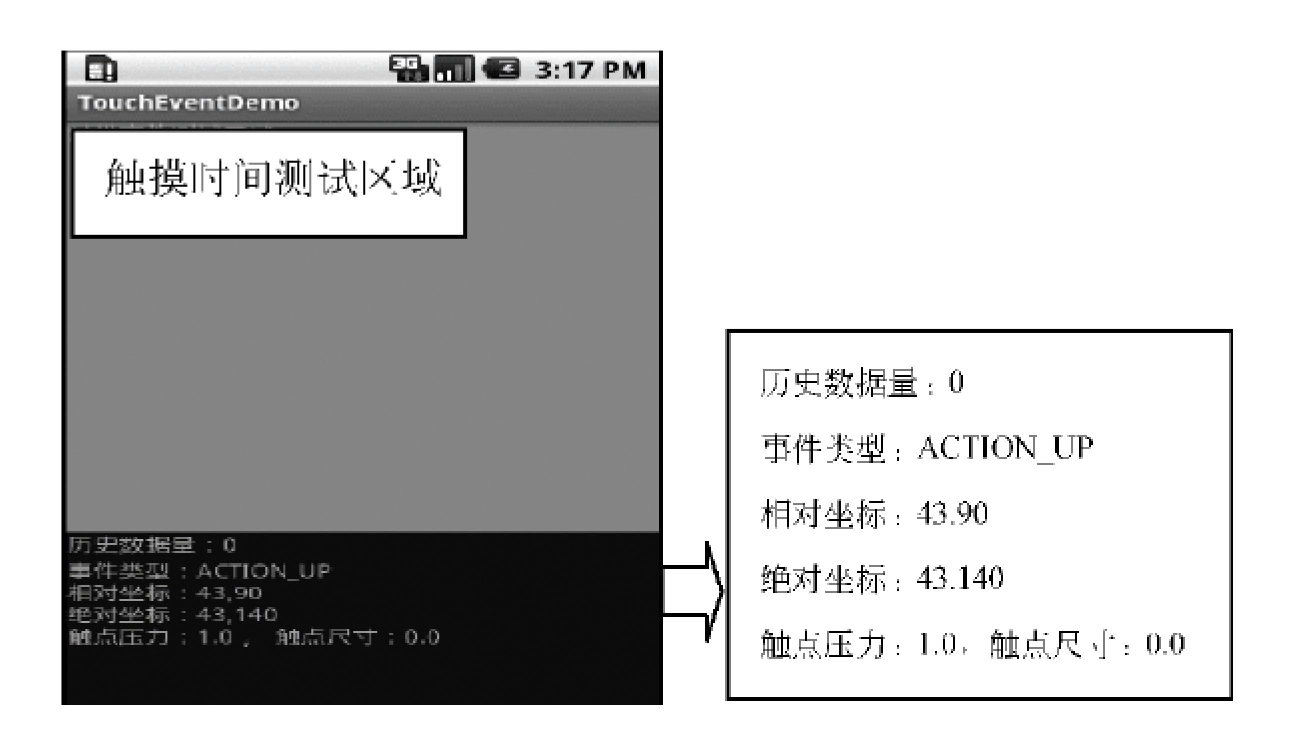


图 5-81　TouchEventDemo 用户界面

在用户界面中使用了线性布局，并加入了 3 个 TextView 控件：

第一个 TextView(ID 为 touch_area)用来标识触摸事件的测试区域；

第二个 TextView(ID 为 history_label)用来显示触摸事件的历史数据量；

第三个 TextView(ID 为 event_label)用来显示触摸事件的详细信息，包括事件类型、相对坐标、绝对坐标、触点压力和触点尺寸。

3）TouchEventDemo 的 XML 文件代码

```
1. <?xml version = "1.0" encoding = "utf - 8"?>
2. < LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
3.   android:orientation = "vertical"
4.   android:layout_width = "fill_parent"
5.   android:layout_height = "fill_parent">
6.   < TextView android:id = "@ + id/touch_area"
7.     android:layout_width = "fill_parent"
8.     android:layout_height = "300dip"
9.     android:background = " # 80A0FF"
10.     android:textColor = " # FFFFFF"
11.     android:text = "触摸事件测试区域">
12.     </TextView >
13.       < TextView android:id = "@ + id/history_label"
14.                          android:layout_width = "wrap_content"
15.     android:layout_height = "wrap_content"
16.     android:text = "历史数据量：" >
17.   </TextView >
18.   < TextView android:id = "@ + id/event_label"
19.     android:layout_width = "wrap_content"
20.     android:layout_height = "wrap_content"
21.     android:text = "触摸事件：" >
22.   </TextView >
23.   </LinearLayout >
```

第 9 行代码定义了 TextView 的背景颜色，# 80A0FF 是颜色代码；第 10 行代码定义了 TextView 的字体颜色。

在代码中为了能够引用 XML 文件中声明的界面元素，使用了下面的代码：

```
TextView labelView = null;
labelView = (TextView)findViewById(R.id.event_label);
TextView touchView = (TextView)findViewById(R.id.touch_area);
final TextView historyView = (TextView)findViewById(R.id.history_label);
```

4）触摸事件的处理

当手指接触到触摸屏、在触摸屏上移动或离开触摸屏时，分别会引发 ACTION_DOWN、ACTION_UP 和 ACTION_MOVE 触摸事件，而无论是哪种触摸事件都会调用 onTouch()函数进行处理。

事件类型包含在 onTouch()函数的 MotionEvent 参数中，可以通过 getAction()函数获取到触摸事件的类型，然后根据触摸事件的不同类型进行不同的处理。

为了能够使屏幕最上方的 TextView 处理触摸事件，需要使用 setOnTouchListener()函数在代码中设置触摸事件监听器，并在 onTouch()函数中添加触摸事件的处理过程代码如下：

```
1. touchView.setOnTouchListener(new View.OnTouchListener(){
2.            @Override
3.            public boolean onTouch(View v, MotionEvent event) {
4.                 int action = event.getAction();
5.                 switch (action) {
6.                 case (MotionEvent.ACTION_DOWN):
7.                     Display("ACTION_DOWN",event);
8.                     break;
9.                case (MotionEvent.ACTION_UP):
10.                     int historySize = ProcessHistory(event);
11.                     historyView.setText("历史数据量: " + historySize);
12.                     Display("ACTION_UP",event);
13.                     break;
14.                 case (MotionEvent.ACTION_MOVE):
15.                     Display("ACTION_MOVE",event);
16.                     break;
17.             }
18.                     return true;
19.               }
20.            });
```

第 7 行代码的 Display()是一个自定义函数，主要用来显示触摸事件的详细信息，函数的代码和含义将在后面进行介绍。

第 10 行代码中的 ProcessHistory()也是一个自定义函数，用来处理触摸事件的历史数据。

第 11 行代码是使用 TextView 显示历史数据的数量。

MotionEvent 参数中不仅有触摸事件的类型信息，还有触点的坐标信息，获取方法是使用 getX()和 getY()函数，这两个函数获取到的是触点相对于父界面元素的坐标信息。如果需要获取绝对坐标信息，则可使用 getRawX()和 getRawY()函数。

触点压力是一个介于 0 和 1 之间的浮点数，用来表示用户对触摸屏施加压力的大小，接

近 0 表示压力较小，接近 1 表示压力较大，获取触摸事件触点压力的方式是调用 getPressure()函数。

触点尺寸指用户接触触摸屏的接触点大小，也是一个介于 0 和 1 之间的浮点数，接近 0 表示尺寸较小，接近 1 表示尺寸较大，可以使用 getSize()函数获取。

Display()将 MotionEvent 参数中的事件信息提取出来，并显示在用户界面上。代码如下：

```
private void Display(String eventType, MotionEvent event){
        int x = (int)event.getX();
        int y = (int)event.getY();
        float pressure = event.getPressure();
        float size = event.getSize();
        int RawX = (int)event.getRawX();
        int RawY = (int)event.getRawY();

        String msg = "";
         msg += "事件类型：" + eventType + "\n";
         msg += "相对坐标：" + String.valueOf(x) + "," + String.valueOf(y) + "\n";
         msg += "绝对坐标：" + String.valueOf(RawX) + "," + String.valueOf(RawY) + "\n";
         msg += "触点压力：" + String.valueOf(pressure) + ", ";
         msg += "触点尺寸：" + String.valueOf(size) + "\n";
         labelView.setText(msg);
      }
```

5）ACTION_MOVE 事件处理

一般情况下，如果用户将手指放在触摸屏上，但不移动，然后抬起手指，应先后产生 ACTION_DOWN 和 ACTION_UP 两个触摸事件。但如果用户在屏幕上移动手指，然后再抬起手指，则会产生这样的事件序列：ACTION_DOWN→ACTION_MOVE→ACTION_MOVE→ACTION_MOVE→……→ACTION_UP。

在手机上运行的应用程序，效率是非常重要的。如果 Android 界面框架不能产生足够多的触摸事件，则应用程序就不能够很精确地描绘触摸屏上的触摸轨迹；如果 Android 界面框架产生了过多的触摸事件，虽然能够满足精度的要求，但却降低了应用程序的效率。

Android 界面框架使用了“打包”的解决方法。在触点移动速度较快时会产生大量的数据，每经过一定的时间间隔便会产生一个 ACTION_MOVE 事件，在这个事件中，除了有当前触点的相关信息外，还包含这段时间间隔内触点轨迹的历史数据信息，这样既能够保持精度，又不至于产生过多的触摸事件。通常情况下，在 ACTION_MOVE 的事件处理函数中，都先处理历史数据，然后再处理当前数据。示例代码如下：

```
private int ProcessHistory(MotionEvent event)
    {
    int historySize = event.getHistorySize();
    for (int i = 0; i < historySize; i++) {
        long time = event.getHistoricalEventTime(i);
        float pressure = event.getHistoricalPressure(i);
```

```
7.          float x = event.getHistoricalX(i);
8.          float y = event.getHistoricalY(i);
9.          float size = event.getHistoricalSize(i);
10.
11.           // 处理过程......
12.     }
13.     return historySize;
14.     }
```

第 3 行代码获取了历史数据的数量,然后在第 4 行至 12 行中循环处理这些历史数据;第 5 行代码获取了历史事件的发生时间;第 6 行代码获取了历史事件的触点压力;第 7 行和第 8 行代码获取历史事件的相对坐标;第 9 行代码获取历史事件的触点尺寸;第 14 行代码返回历史数据的数量,主要用于界面显示。

Android Emulator 并不支持触点压力和触点尺寸的模拟,所有触点压力恒为 1.0,触点尺寸恒为 0.0;同时 Android Emulator 上也无法产生历史数据,因此历史数据量一直显示为 0。

5.5 Android 游戏编程:Tank 大战

现在开始真正地编写 Android 程序。本节不再叙述大段的原理,而是在编程过程中渗透对原理、概念的讲解。

5.5.1 创建程序 Hello Tank

(1) 首先打开 Eclipse,选择菜单中的 File→New→Project。如图 5-82 所示。

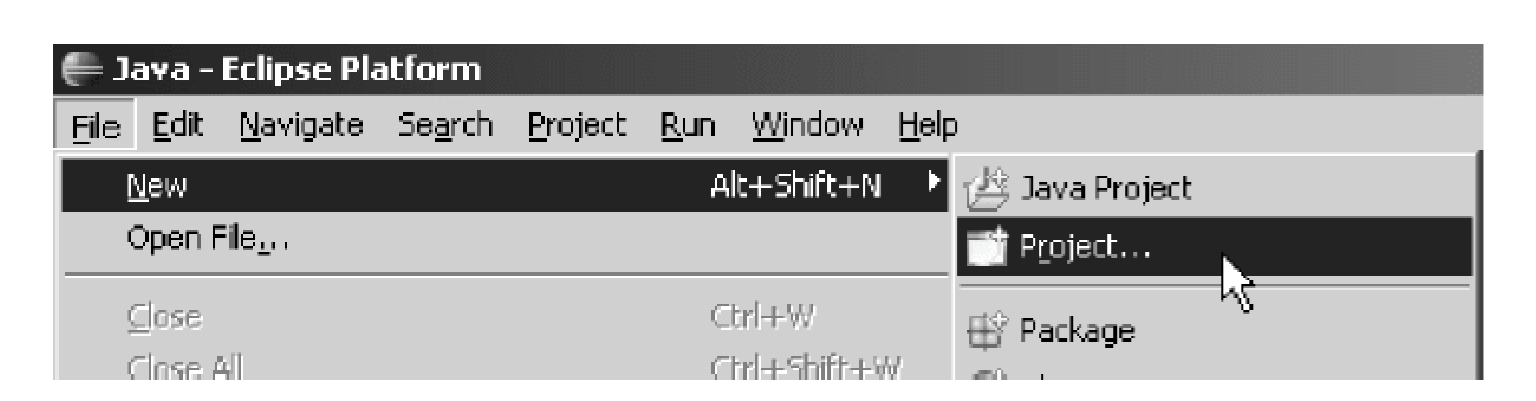


图 5-82 打开 eclipse

(2) 选择 Android→Android Project。如图 5-83 所示。

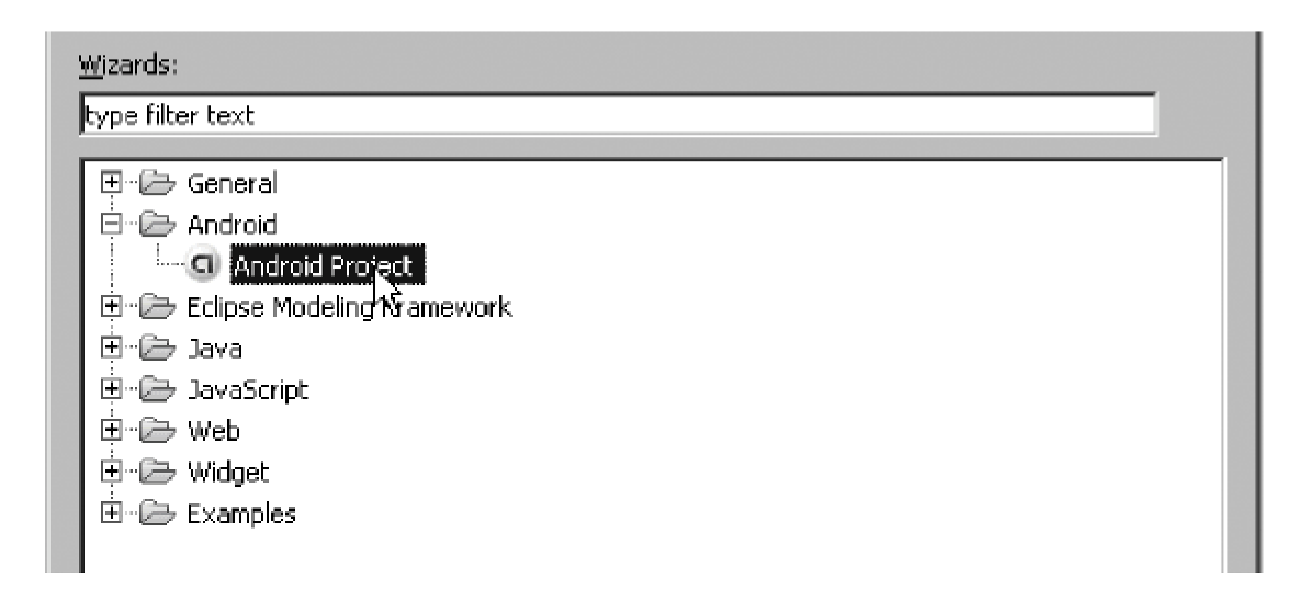


图 5-83 选择 Android Project

(3) 输入项目的信息,因为要复刻经典游戏坦克大战,所以程序就取名 Tank。如图 5-84 所示。

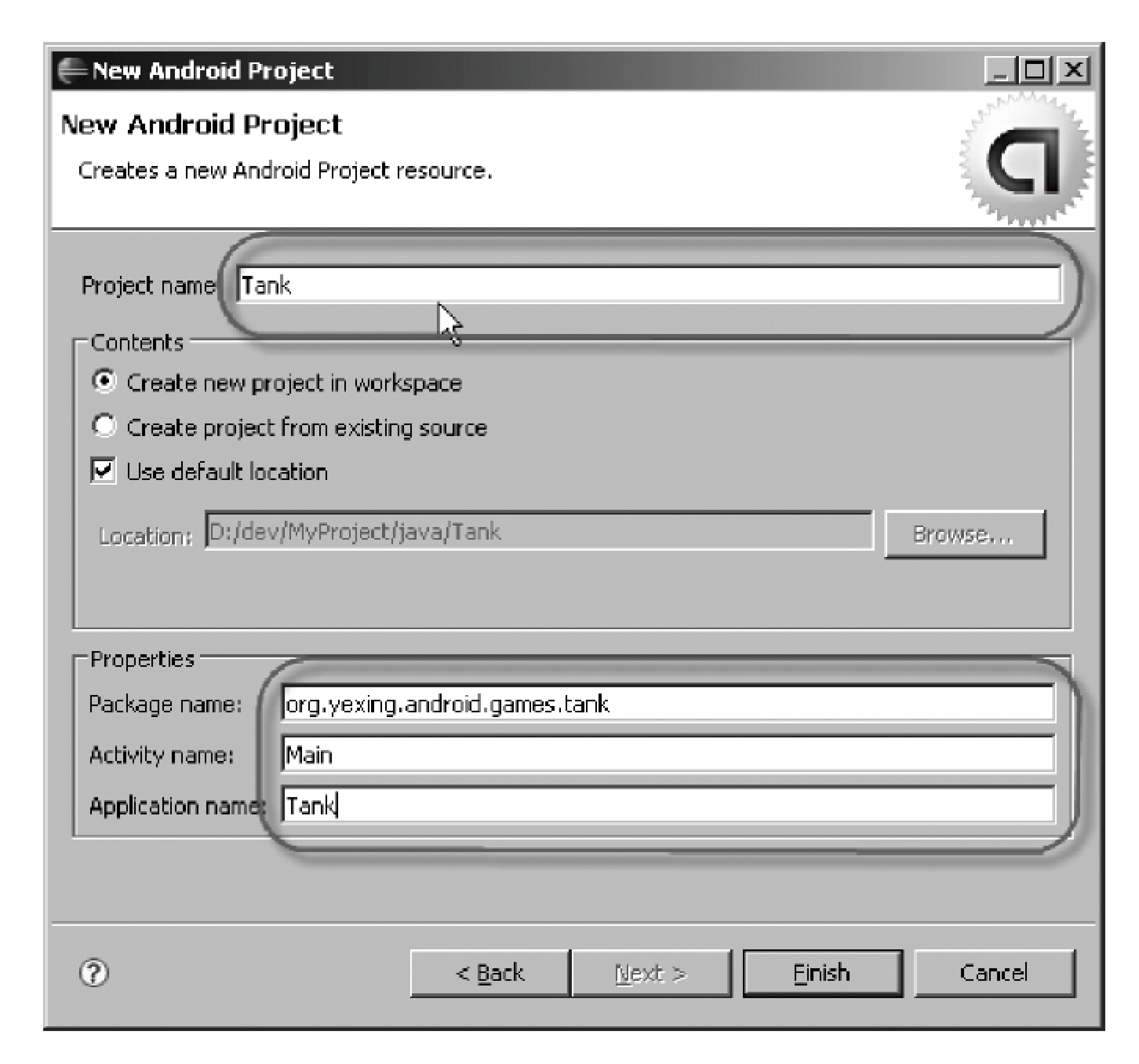


图 5-84 输入 Tank 的项目信息

现在,一个 Android 项目就创建完成了,可以在 Eclipse 的 Package Explorer 中看到所创建的项目了。如图 5-85 所示。

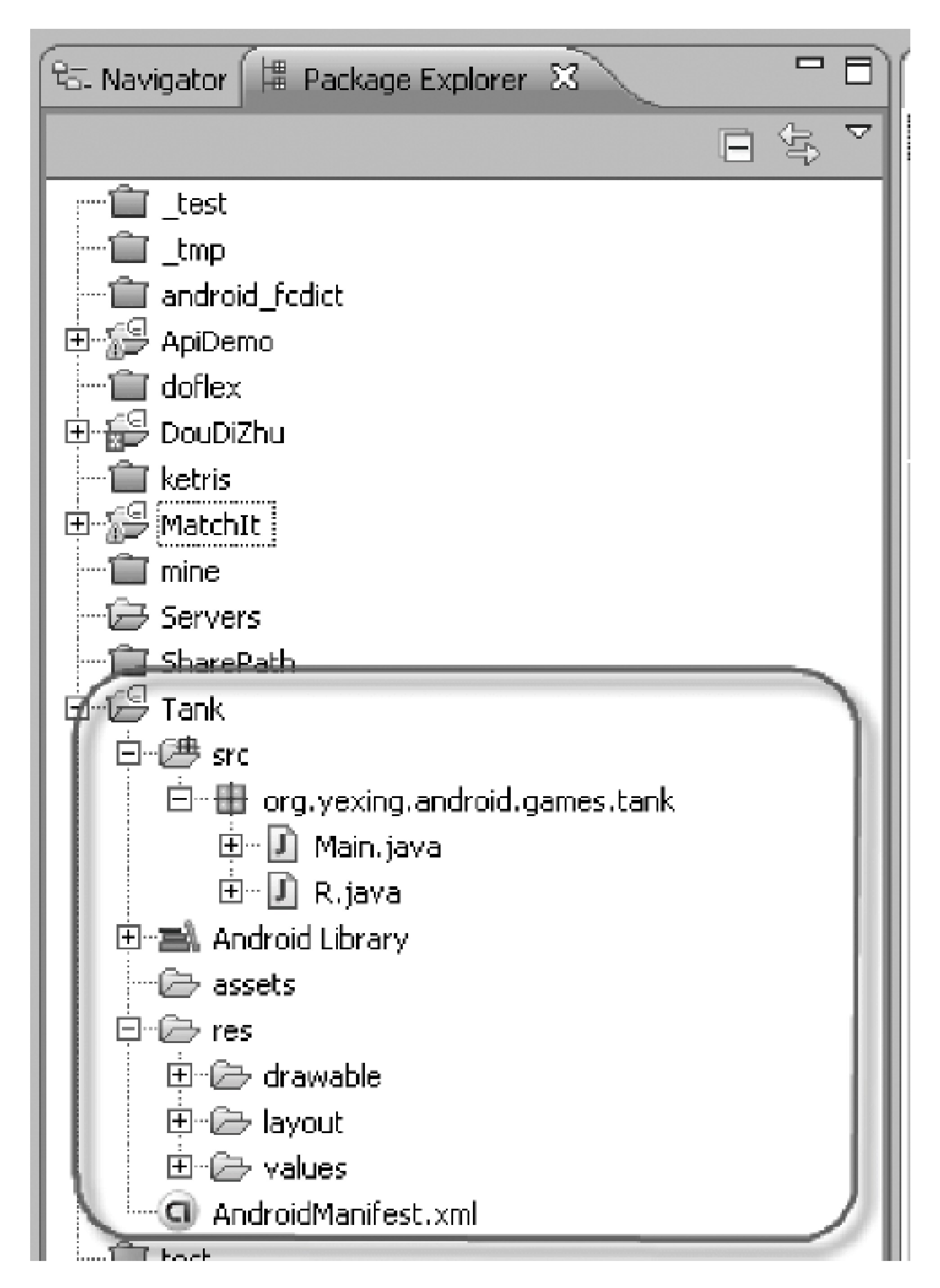


图 5-85 在 Eclipse 的 Package Explorer 下查看所创建的项目

由于有 ADT，虽然只输入了几个名字，但这个项目实际上已经可以运行了。单击项目名，选择 Run As→Android Application。如图 5-86 所示。

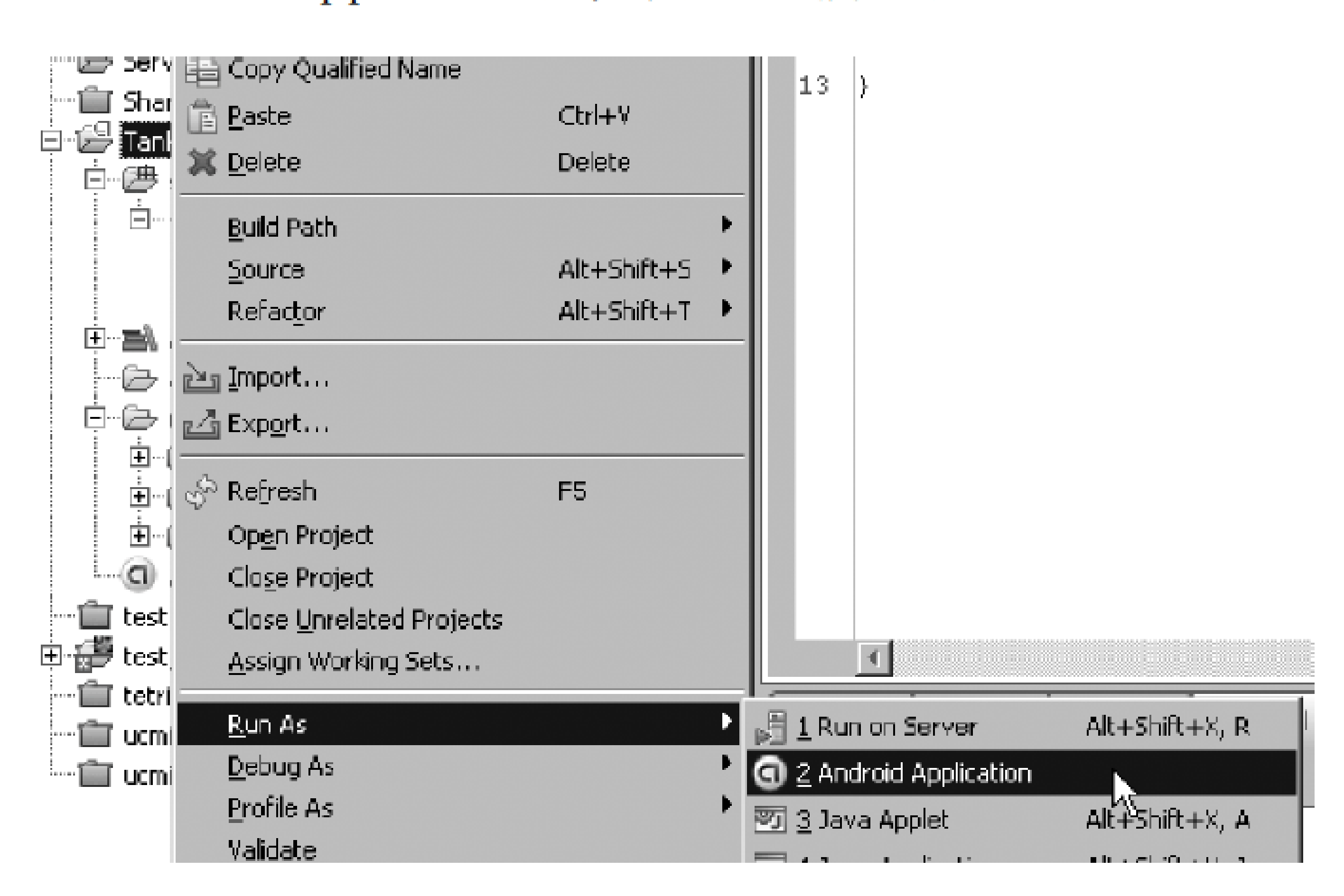


图 5-86　Tank 项目的运行

不出意外的话，就会看到一个手机模拟器被启动，而刚刚建立的程序会被运行起来。如图 5-87 所示。

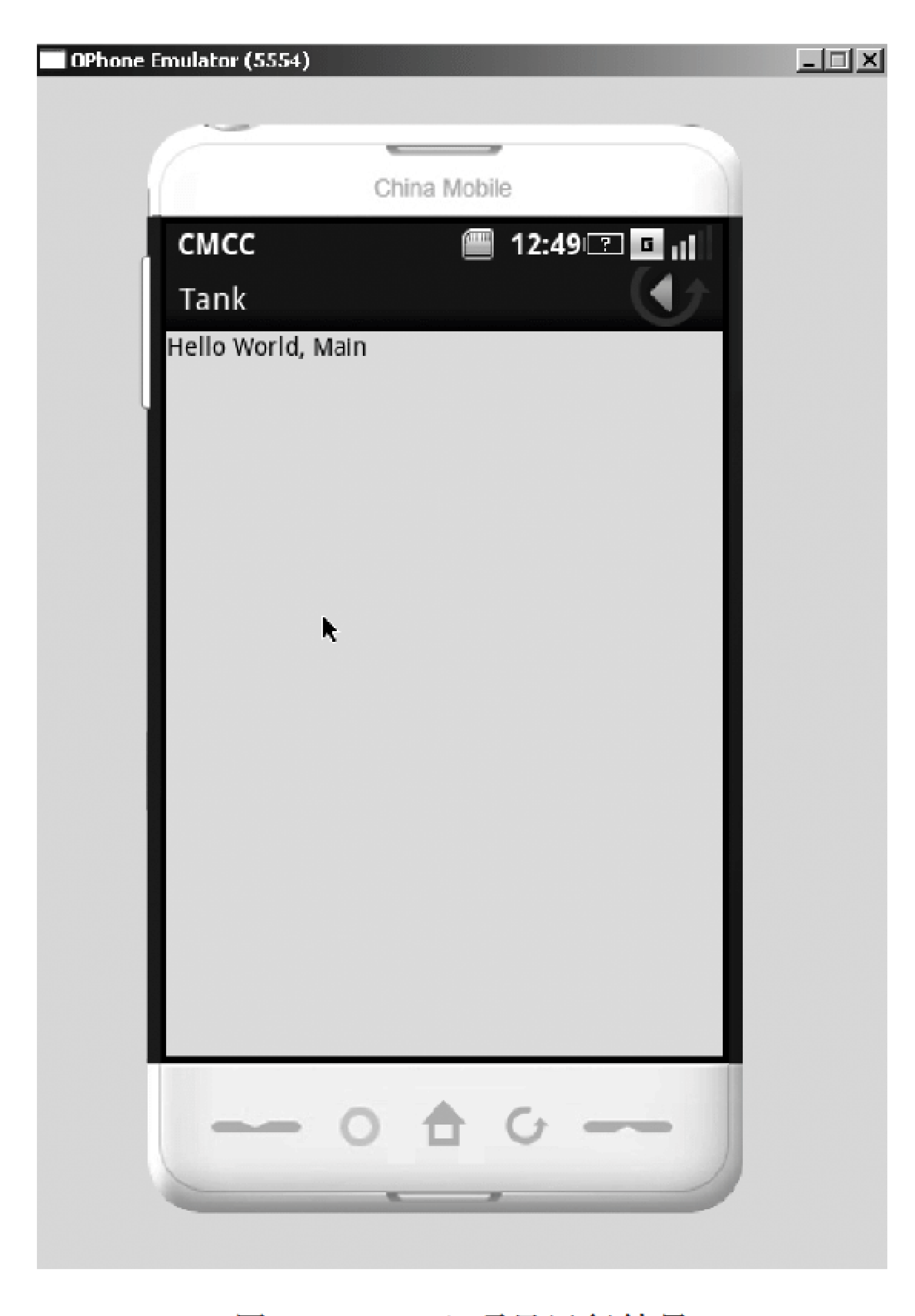


图 5-87　Tank 项目运行结果

如果发现模拟器启动了，而程序并没有被运行，可能需要手工启动程序。这里需要用到一个重要的工具 DDMS(Davlik Debug Manager)。运行 DDMS 快捷方法是单击 eclipse 右上角的 Open Perspective，如果在弹出的列表中没有 DDMS，那么单击 Others，如图 5-88 所示。

选择 DDMS，如图 5-89 所示。

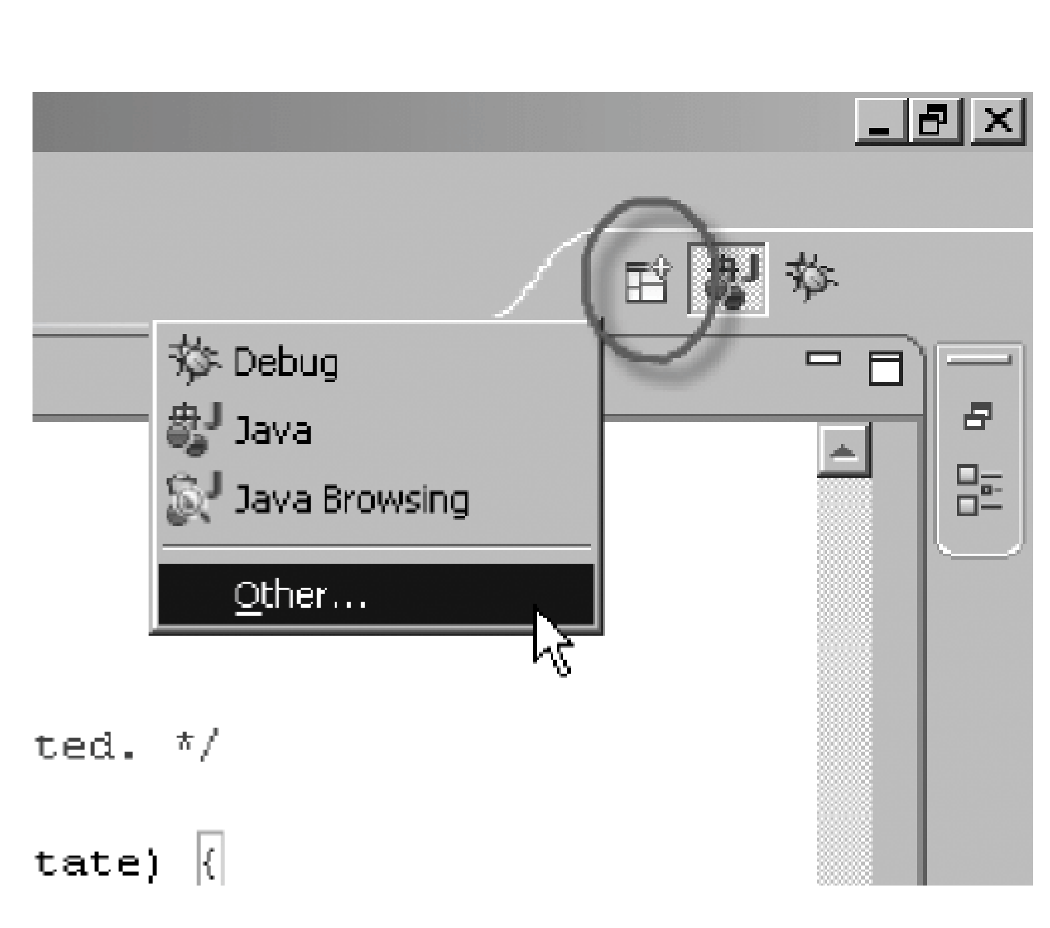




图 5-88　查找 DDMS

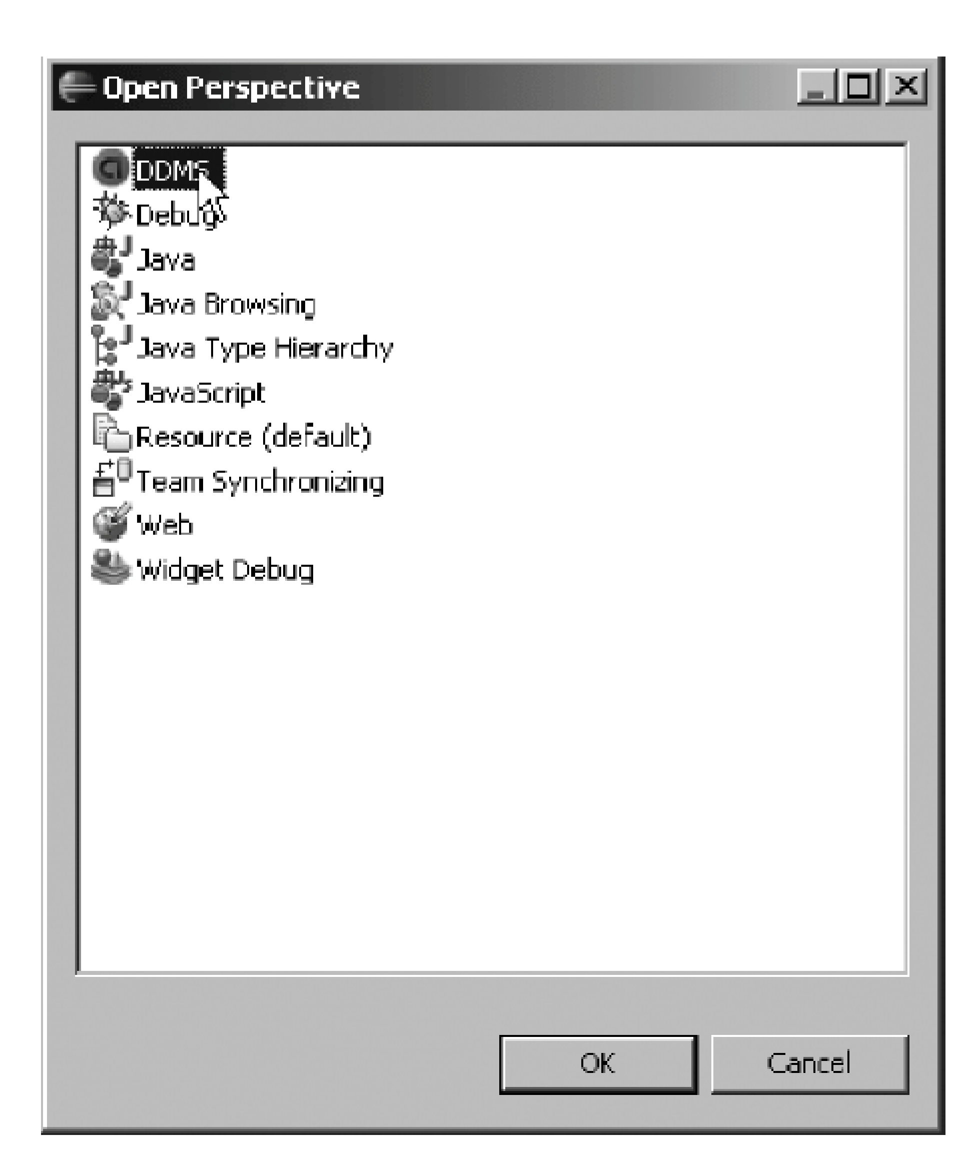


图 5-89　选择 DDMS

这样就打开了 DDMS 界面，这个工具以后会经常用到，如图 5-90 所示。

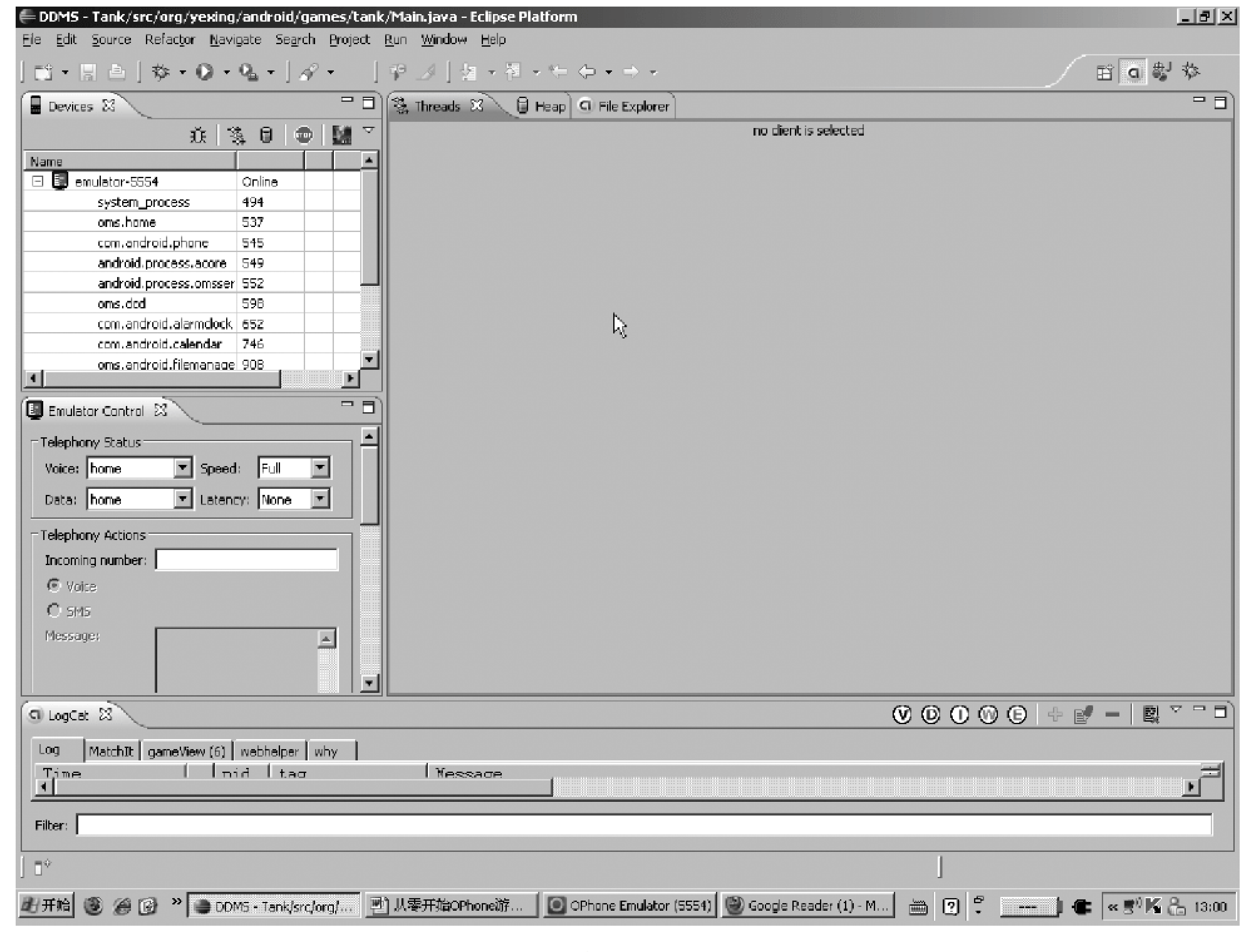


图 5-90　打开 DDMS 界面

刚刚说到模拟器启动了而程序并没有被运行，很可能是在模拟器启动过程中 DDMS 失去了与模拟器的链接。解决方法很简单：

(1) 单击 Devices 标签下的工具栏，选择 Reset adb，如图 5-91 所示。

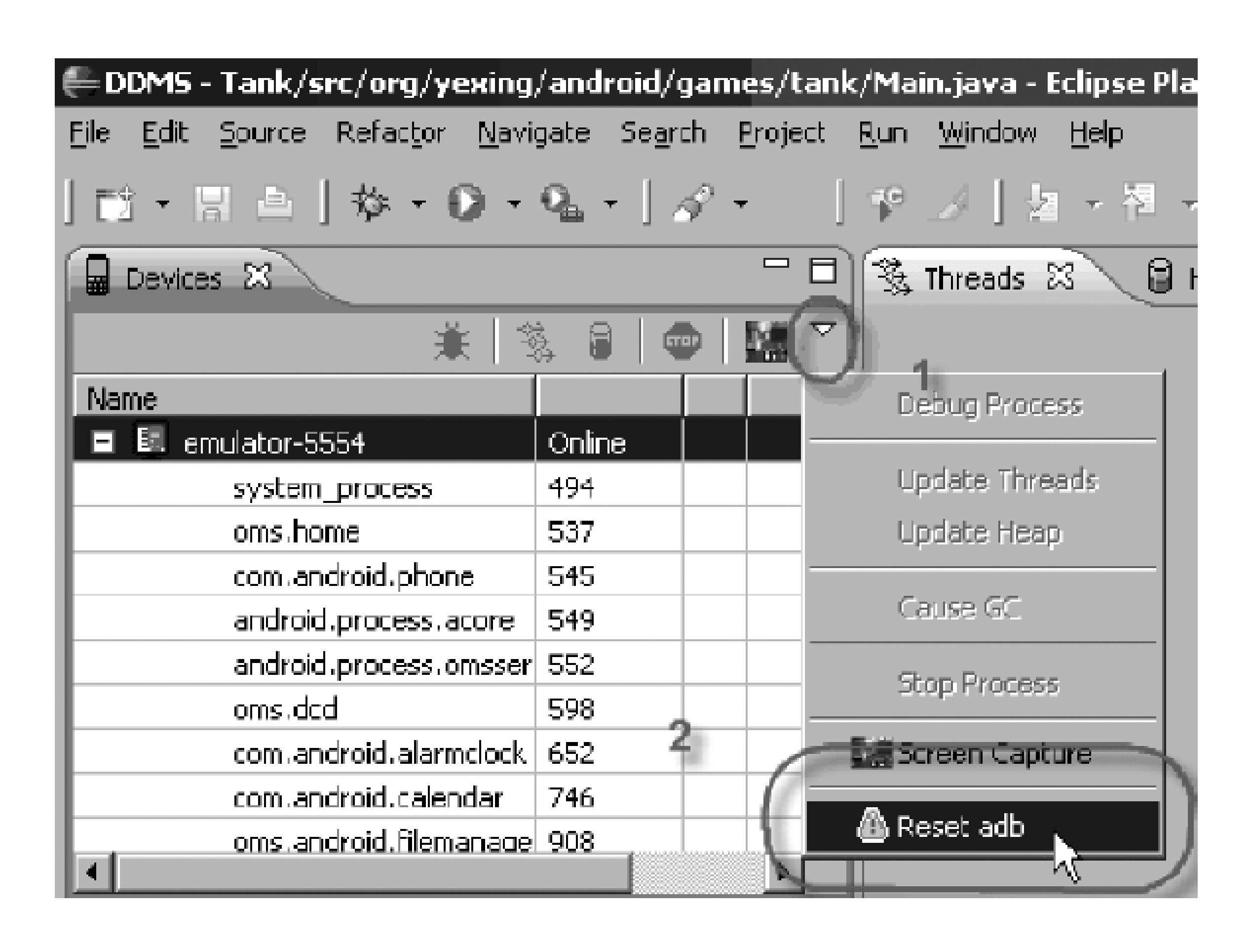


图 5-91 选择 Reset adb

(2) 然后单击项目名称，Run As→Android Application。

除了单击运行项目，还可以通过工具栏上的运行按钮启动程序，如图 5-92 所示。

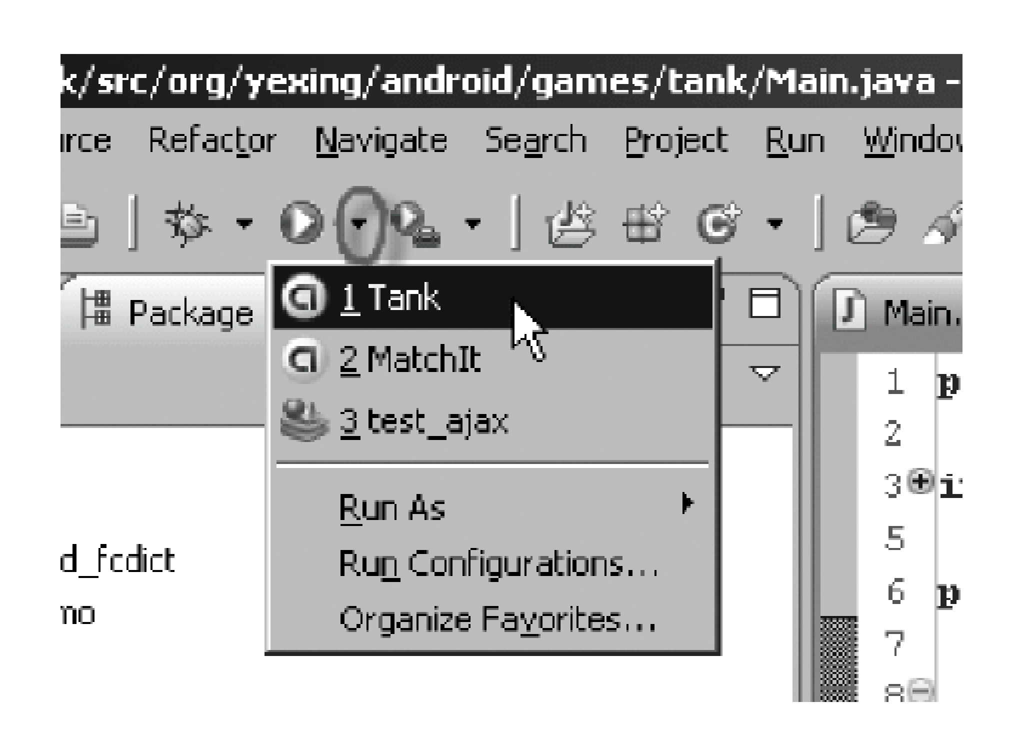


图 5-92 通过运行按钮启动程序

位于运行按钮左边的是 Debug 按钮，这两个按钮以后也会经常用到。

现在已经有了第一个可以运行的 Android，虽然可能对 ADT 生成的一堆文件感到一头雾水，也不知道程序界面上"Hello World, Main"是从哪里来的，但是没关系，随着本书的深入介绍，就会逐渐熟悉 Android 项目的目录结构，程序设计的原则和方法，以及调试和部署的方法。

5.5.2 显示文字和图片

从本节开始，读者就要编写代码了。按照作者的原则——少一些理论，多一些实践，代码中可能会有跳跃的地方。但是请大家不要着急，随着学习的深入，很快就会了解其中的奥秘。不过在开始之前，需要先来理顺一下思路，看看完成一个坦克大战游戏需要哪些工作：首先，需要一个基本的程序，这个程序能够在 Android 上运行；这个程序要能够显示图形

包括地图，主角和 NPC 等；程序能够接收用户的输入，控制主角移动；程序要能够控制 NPC 和子弹的移动；程序还能对各种事件做出判断，比如击中敌人，获得物品，胜利或者失败。

现在就从基本程序开始，一步一步实现它。

首先，看一下刚刚生成的文件目录，如图 5-93 所示。



图 5-93　生成的文件目录

在源文件目录下，只有 Main.java 和 R.java 两个文件，刚刚被命名成 Main.java 的文件就是程序的入口文件。而 R.java 是由插件来维护的资源定义文件，可以先不管它。

Main.java 内容如下：

```
package org.yexing.android.games.tank;
import android.app.Activity;
import android.os.Bundle;
public class Main extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

很幸运，Main.java 的代码非常之少，而且还有一段注释，以至于很容易知道函数 onCreate 的作用，需要解释的只是 setContentView()。先不要管注释中提到的 Activity 和 setContentView 的参数 R.layout.main，可以使用 setContentView 的另一种形式：setContentView(View view)。setContentView 的作用是设定当前使用的视图即 View(依此理解，可以有很多个 View，需要用哪个就可以把它作为 setContentView 的参数显示出来)。View 是一个非常重要的组件，它可以用来显示文字、图片，也可以接收客户的操作，比如触摸屏、键盘等，而游戏中正是需要绘图和交互，看来 View 很符合我们的需要(但是请注意，使用 View 并不是需要的最终方案，原因会在后面说明。此处介绍 View 是为了讲解基础的图形和用户控制)。

下面就要订制一个属于自己的 View，可以通过继承自系统提供的 View，并重载相关的函数来实现。创建类的方法如下：

单击包名 New→Class，如图 5-94 所示。

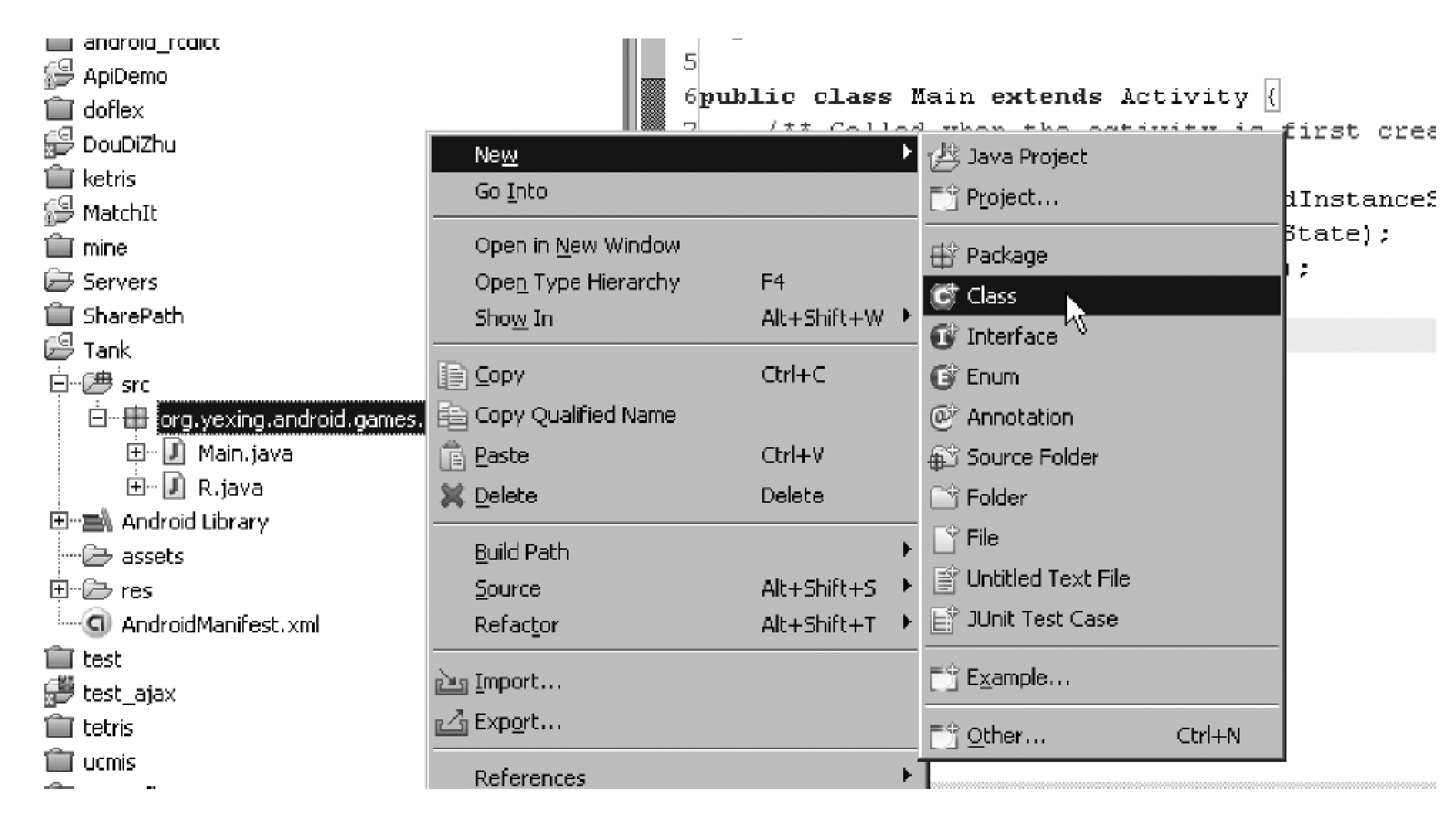


图 5-94　创建类

同时将这个 View 类命名为 GameView，并且由 android. view. View 继承，如图 5-95 所示。

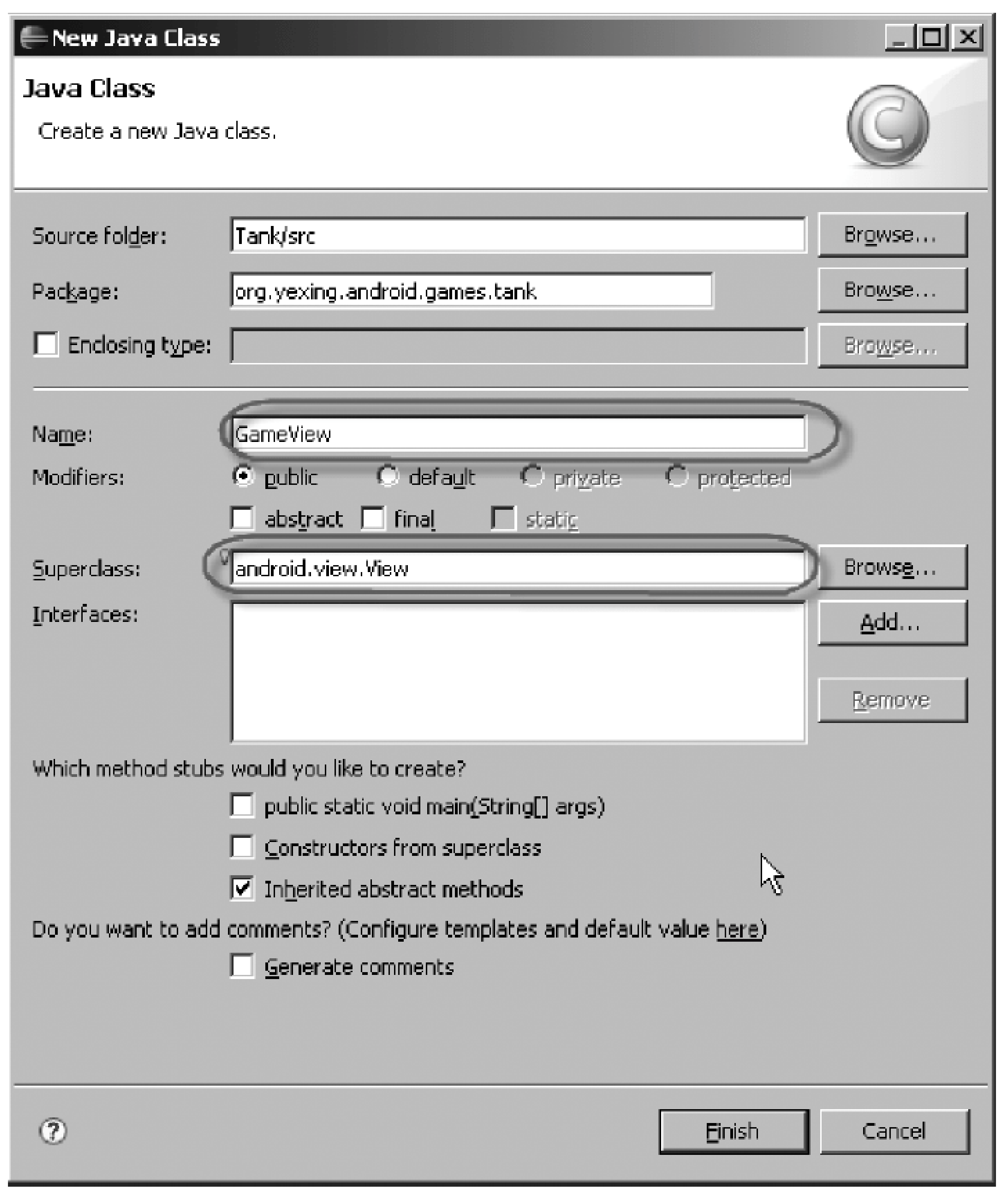


图 5-95　设置类的相关信息

单击 Finish 按钮，一个 View 类就创建好了。这里是第一次创建类，以后就不会有图片演示了，请大家记住这个方法。GameView 创建好了，但是代码还有一些错误，这里介绍一下 eclipse 的使用技巧，将鼠标悬停在有错误的位置，或者将光标停在有错误的行，然后按 Ctrl+1 键，就会出现修改建议，大多数时候，使用修改建议都可以改正错误，如图 5-96 所示。

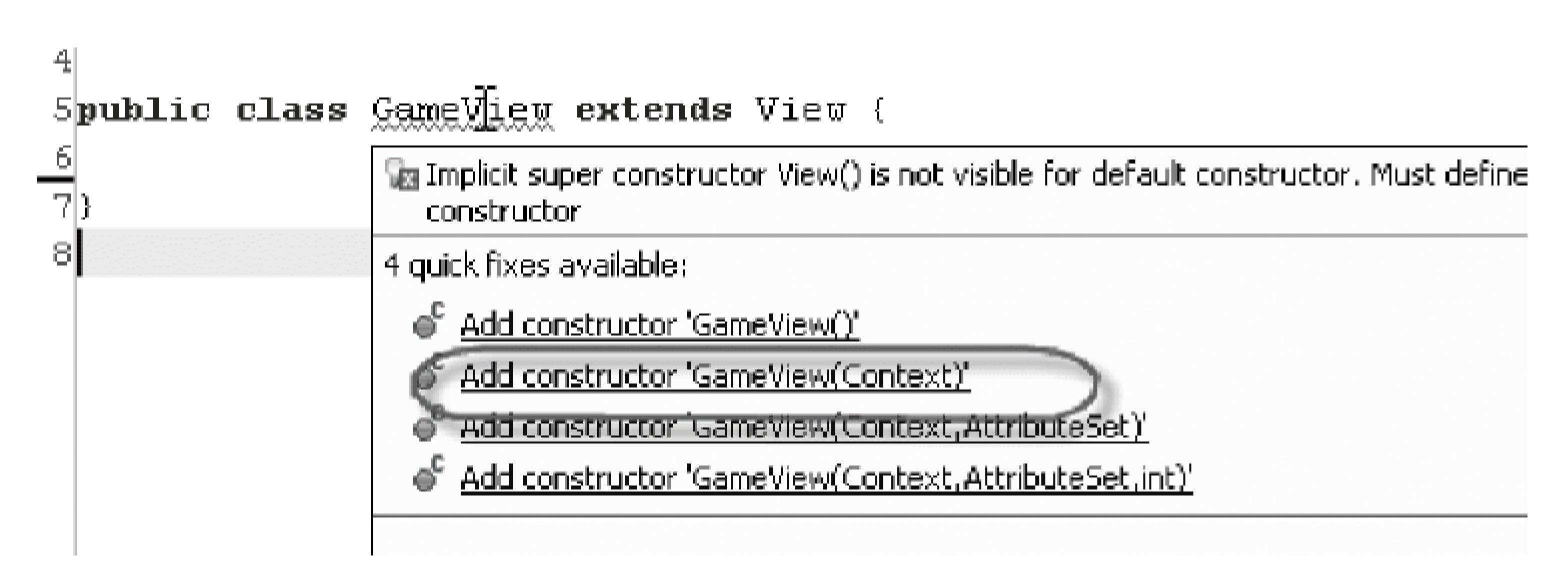


图 5-96　使用修改建议改正错误

可以看出来，刚刚的错误是因为没有创建构造函数，选择修改建议的第二项，增加一个构造函数。

```
public GameView(Context context) {
    super(context);
    // TODO Auto - generated constructor stub
}
```

至此 View 就创建好了。

回到 Main.java，刚刚说了，只要将 View 作为 setContentView 的参数，这个 View 就可以被显示出来：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(new GameView(this));
}
```

现在运行模拟器，看看程序变成什么样子了，如图 5-97 所示。

不要意外，屏幕上就是一片空白，因为虽然创建了一个 View，但是没有让它显示任何内容。下面我们就会在 View 上显示一段文字和一张图片。

让 View 显示内容也很简单，只需要重载 View 的 onDraw 函数，把相应的语句写入 onDraw 中即可。打开 GameView.java，单击菜单 Source→Override/Implement Method，如图 5-98 所示。

选中 onDraw，单击 OK 按钮，如图 5-99 所示。

下面这段代码就会被加入到程序当中，所有与显示有关的代码都会在这里面完成：

```
@Override
protected void onDraw(Canvas canvas) {
    // TODO Auto - generated method stub
    super.onDraw(canvas);
}
```

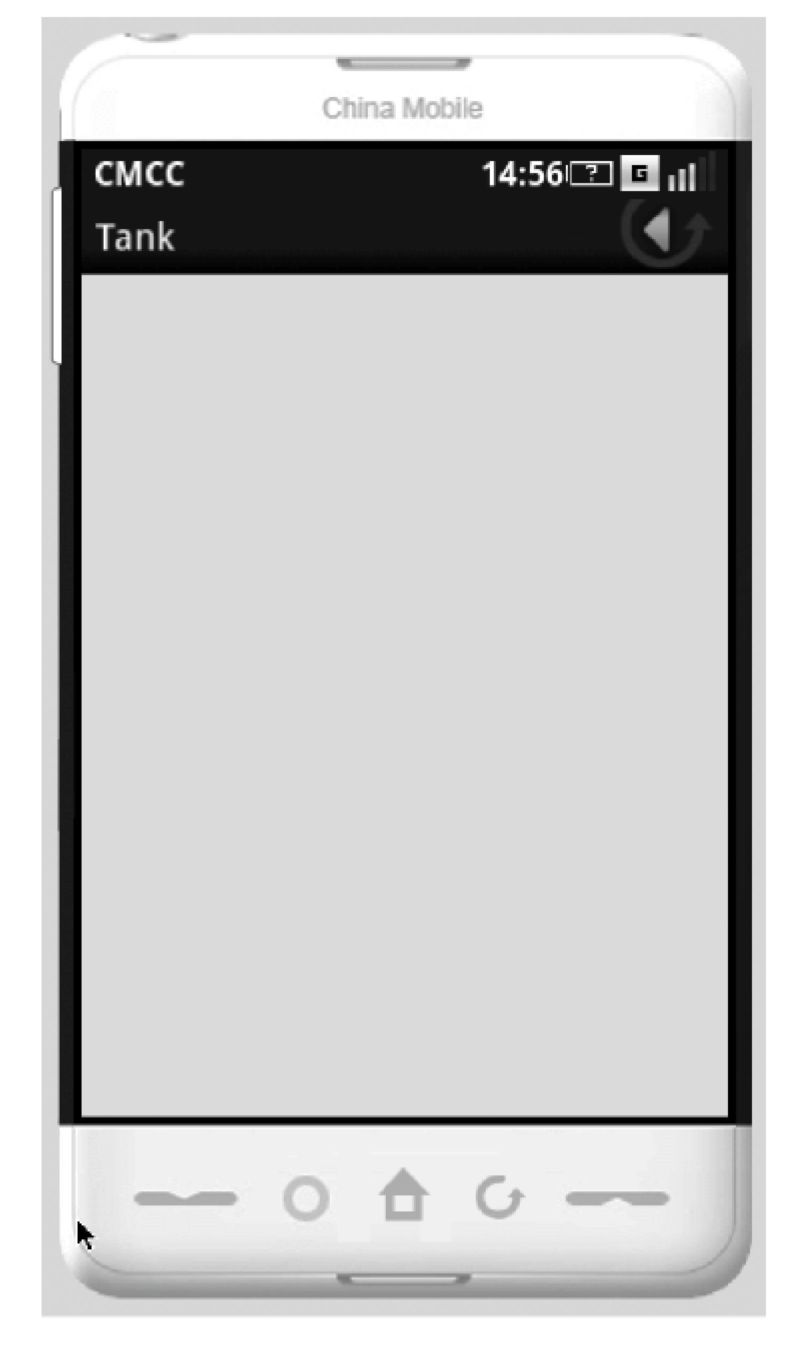


图 5-97　运行模拟器

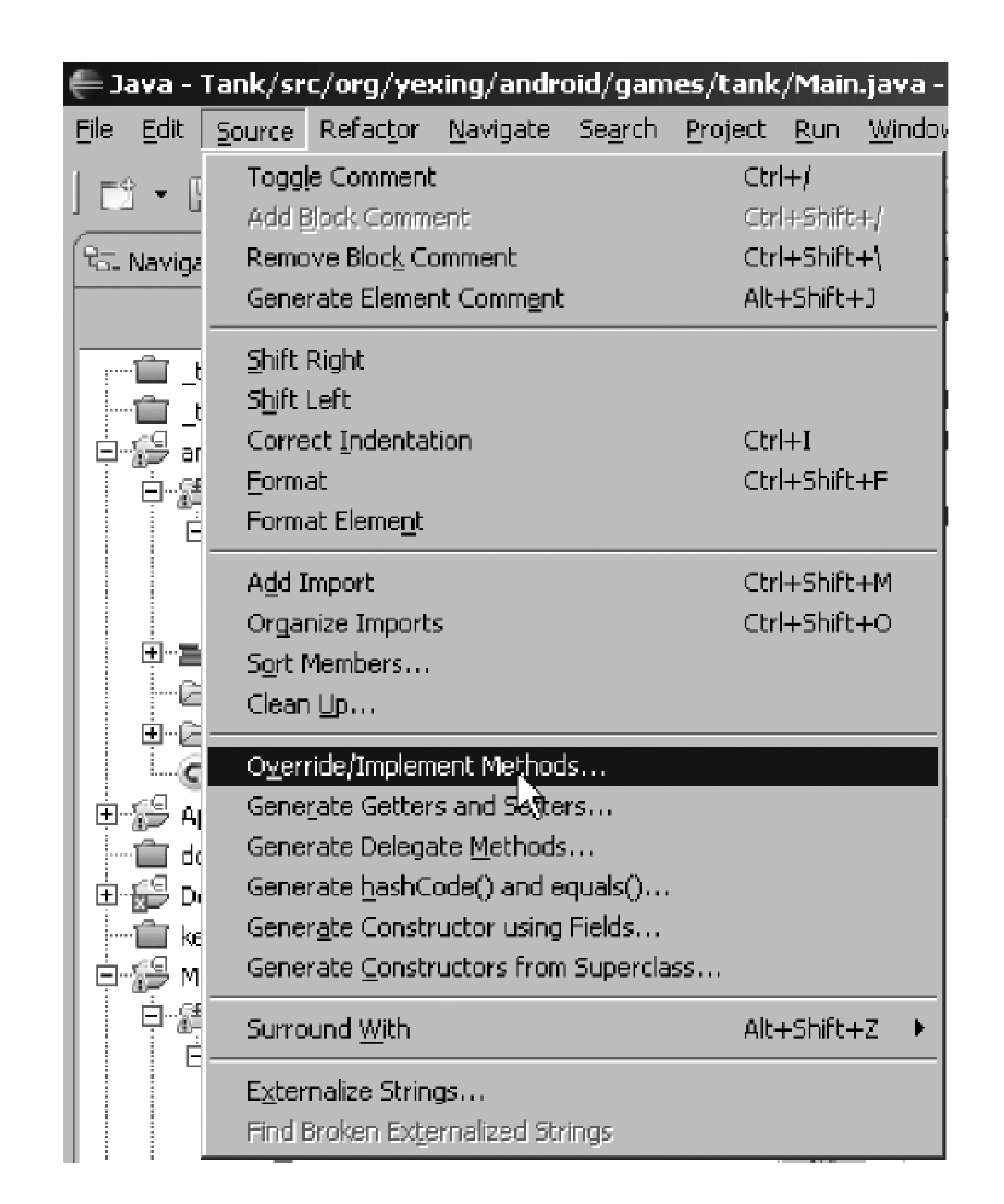


图 5-98　View 内容显示方法

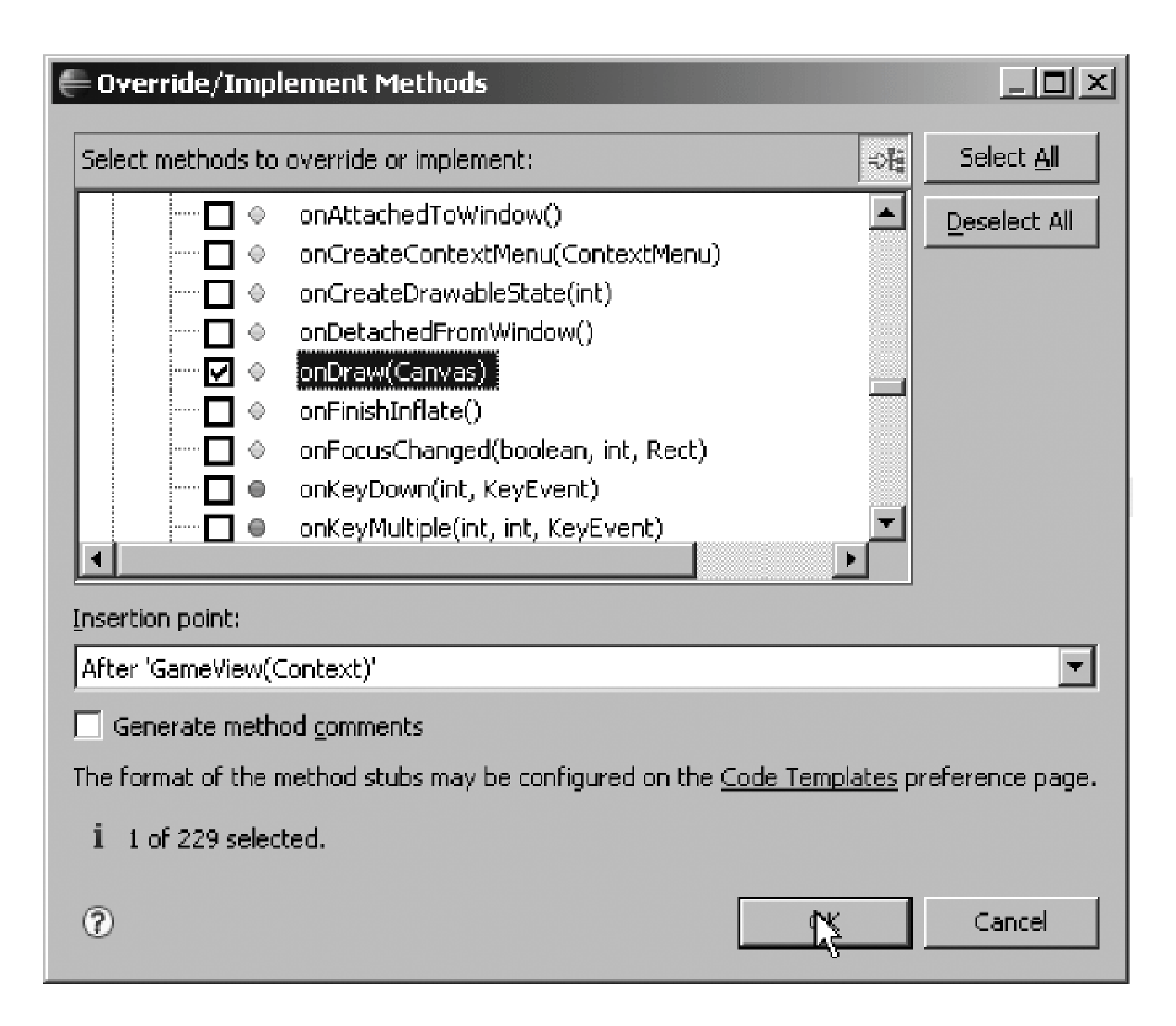


图 5-99　onDraw 的选择

这里遇到了又一个非常重要的类 Canvas，Canvas 一般翻译成画布，所有的绘图操作都是通过 Canvas 中的函数来完成的，比如显示文字的函数 Canvas. drawText()，显示位图的函数 Canvas. drawBitmap()，以及各种绘制图形的函数如 Canvas. drawRect()，Canvas. drawArc()等。下面显示一段文字在屏幕上，如图 5-100 所示。

```
protected void onDraw(Canvas canvas) {
    // TODO Auto - generated method stub
    super.onDraw(canvas);
    canvas.drawText("坦克大战", 50, 50, new Paint());
}
```



图 5-100　模拟器上"坦克大战"的显示

坦克大战四个字已经出现在了屏幕上。让我们来详细看一下这条语句：

```
canvas.drawText("坦克大战", 50, 50, new Paint());
```

第一个参数是要显示的文字，第二、第三个参数是文字在屏幕上的坐标，说到坐标得多讲两句。在 2D 编程中，屏幕坐标的原点是屏幕的左上角，横向向右增大，纵向向下增大，如上图所示。最后一个参数是 Paint，通常翻译成画笔，它决定了文字或图形的颜色、字体、线条粗细等，后面用到相应属性的时候会详细介绍。那么这条语句就是在屏幕上(50，50)的位置用默认的画笔写出"坦克大战"四个字。另外如果 eclipse 提示代码错误，不要忘了用 Ctrl＋1 键。

有了文字，下面就是图像了。显示图像比显示文字略微复杂一些，首先需要准备一张位图，图片必须是 png 格式的，文件名只能是小写字母，数字和下划线，如图 5-101 所示。

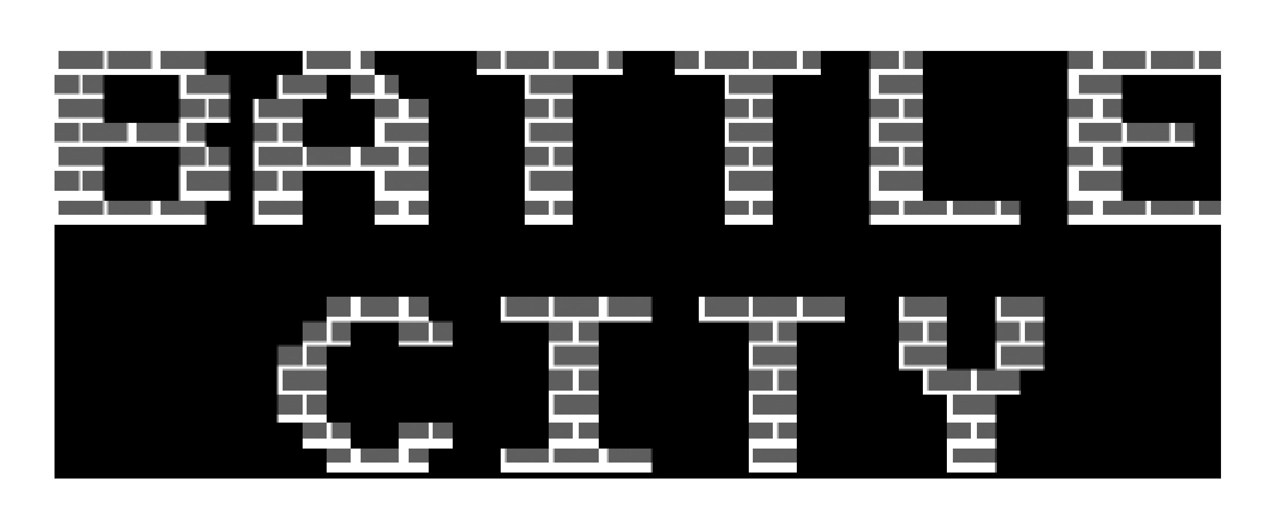


图 5-101　battlecity.png 位图

然后将这张图片复制到工程的 res/drawable 目录下。可以直接在 eclipse 的目录树中粘贴，如图 5-102 所示。

显示位图的函数是 Canvas.drawBitmap()，drawBitmap 有很多种形态，先看其中最简

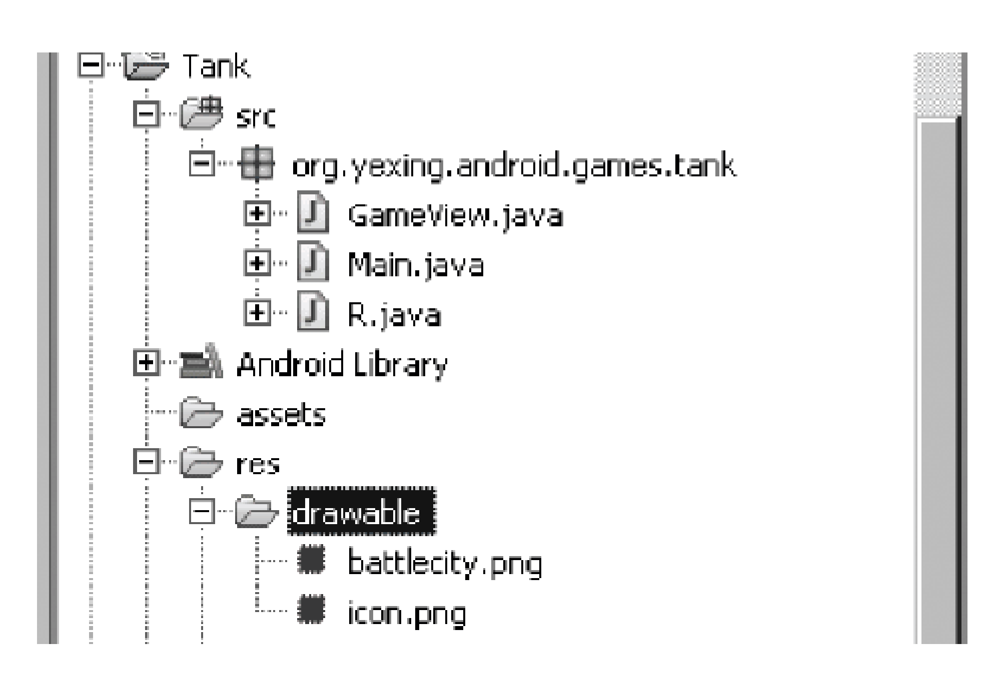


图 5-102　位图导入方法

单的一种。

```
canvas.drawBitmap(bitmap, left, top, paint)
```

乍一看似乎和 drawText 差不多，4 个参数有 3 个都相同，但这第一个参数 bitmap 要比文本复杂得多。首先，他是一个 Bitmap 类实例，因为现在还不需要这个类的其他功能，所以不过多介绍 Bitmap，只考虑它是怎么来的。得到 Bitmap 实例的方法也有很多种，这里只介绍其中的一种：

```
BitmapFactory.decodeResource(res, id);
```

此方法可以返回一个 bitmap 实例，但是这个函数还需要两个参数 res 和 id。res 是 Resources 实例，而 id 是一个整数，下面分别了解这两个参数。res 的地位跟 bitmap 差不多，只需要作为参数被使用，因此，只要得到实例就可以了，获得 Resources 实例的方法如下：

```
res = context.getResources();
```

事情似乎越来越复杂了，因为这段代码里面又多了一个陌生面孔 context。context 是 Context 实例，Context 通常翻译做上下文，这个名称似乎有点晦涩，它究竟是什么呢？回头看看写好的程序：

```
public GameView(Context context) {
    super(context);
    // TODO Auto - generated constructor stub
}
```

这时候有一个 context 实例，继续朔源而上，在 Main.java 中：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(new GameView(this));
}
```

原来，context 指向 Main 类。好了，终于找到 res 的源头了。还有另外一个分支第二个参数 id。

```
BitmapFactory.decodeResource(res, id);
```

id 是一个整形，它到底是谁的 id 呢？还是需要往前面找，还记得第一次见到函数 setContentView 时是什么样子吗？

```
setContentView(R.layout.main);
```

对，它的参数是 R.layout.main，后来被替换成了 GameView 实例。R.layout.main 就是一个整数。它被定义在文件 R.java 中，前面讲过 R.java 是由插件维护的资源定义文件。说到这里大家应该猜到了吧。现在打开 R.java 文件：

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int battlecity = 0x7f020000;
        public static final int icon = 0x7f020001;
    }
    public static final class layout {
        public static final int main = 0x7f030000;
    }
    public static final class string {
        public static final int app_name = 0x7f040001;
        public static final int hello = 0x7f040000;
    }
}
```

果然，位图文件 battlecity.png 在这里面也被分配了一个 id：R.drawable.battlecity，没错，就是它了，这就是要找的 id。至此为止，终于可以使用 drawBitmap 了。

对于一次创建，多次使用的资源，将其放到构造函数里面。增加了图形显示的 GameView 如下：

```
public class GameView extends View {

    Bitmap bmp;
    public GameView(Context context) {
        super(context);
        // TODO Auto-generated constructor stub
        Resources res = context.getResources();
        bmp = BitmapFactory.decodeResource(res, R.drawable.battlecity);
    }

    @Override
    protected void onDraw(Canvas canvas) {
```

```
            // TODO Auto-generated method stub
            super.onDraw(canvas);
            canvas.drawText("坦克大战", 0, 50, new Paint());
            canvas.drawBitmap(bmp, 0, 100, new Paint());
        }
    }
```

运行效果如图 5-103 所示。



图 5-103　坦克大战运行结果

第6章　基于 Android 的无线传感网络

本章主要深入介绍基于 Android 的无线传感网络，讲解无线传感网络在 Android 系统上的开发过程及开发方法。通过本章学习，读者应该掌握以下内容：

（1）Android 中的传感器。

（2）系统总体介绍。

（3）系统模块介绍。

（4）Android 在 WSN 中的应用现状和前景。

（5）无线传感器网络的应用实例。

（6）WSN 的安全性问题。

6.1　Android 中的传感器

自从苹果公司在 2007 年发布第一代 iPhone 以来，以前看似和手机挨不着边的传感器也逐渐成为手机硬件的重要组成部分。如果读者使用过 iPhone、HTC Dream、HTC Magic、HTC Hero 及其他的 Android 手机，会发现通过将手机横向或纵向放置，屏幕会随着手机位置的不同而改变方向。这种功能就需要通过重力传感器来实现，除了重力传感器，还有很多其他类型的传感器被应用到手机中，例如磁阻传感器就是最重要的一种传感器。虽然手机可以通过 GPS 来判断方向，但在 GPS 信号不好或根本没有 GPS 信号的情况下，GPS 就形同虚设。这时通过磁阻传感器就可以很容易判断方向（东、南、西、北）。有了磁阻传感器，也使罗盘（俗称指向针）的电子化成为可能。在 Android 应用程序中使用传感器要依赖于 android.hardware.SensorEventListener 接口。通过该接口可以监听传感器的各种事件。SensorEventListener 接口的代码如下：

```
    package android.hardware;
    public interface SensorEventListener {
    public void onSensorChanged(SensorEvent event);
    public void onAccuracyChanged(Sensor sensor, int accuracy);
}
```

在 SensorEventListener 接口中定义了两个方法：onSensorChanged 和 onAccuracyChanged。当传感器的值发生变化时，例如磁阻传感器的方向改变时会调用 onSensorChanged 方法。当传感器的精度变化时会调用 onAccuracyChanged 方法。

onSensorChanged 方法只有一个 SensorEvent 类型的参数 event，其中 SensorEvent 类

有一个 values 变量非常重要，该变量的类型是 float[]。但该变量最多只有 3 个元素，而且根据传感器的不同，values 变量中元素所代表的含义也不同。

在解释 values 变量中元素的含义之前，先来介绍一下 Android 的坐标系统是如何定义 X、Y、Z 轴的。

X 轴的方向是沿着屏幕的水平方向从左向右。如果手机不是正方形的话，较短的边需要水平放置，较长的边需要垂直放置。

Y 轴的方向是从屏幕的左下角开始沿着屏幕的垂直方向指向屏幕的顶端。

将手机平放在桌子上，Z 轴的方向是从手机里指向天空。下面是 values 变量的元素在主要的传感器中所代表的含义。

6.1.1 方向传感器

在方向传感器中 values 变量的 3 个值都表示度数，它们的含义如下：

values[0]：该值表示方位，也就是手机绕着 Z 轴旋转的角度。0 表示北(North)；90 表示东(East)；180 表示南(South)；270 表示西(West)。如果 values[0]的值正好是这 4 个值，并且手机水平放置，表示手机的正前方就是这 4 个方向。可以利用这个特性来实现电子罗盘，实例 76 将详细介绍电子罗盘的实现过程。

values[1]：该值表示倾斜度，或手机翘起的程度。当手机绕着 X 轴倾斜时该值发生变化。values[1]的取值范围是 $-180 \leqslant$ values[1] $\leqslant 180$。假设将手机屏幕朝上水平放在桌子上，这时如果桌子是完全水平的，values[1]的值应该是 0(由于很少有桌子是绝对水平的，因此，该值很可能不为 0，但一般都是 -5 和 5 之间的某个值)。这时从手机顶部开始抬起，直到将手机沿 X 轴旋转 180°(屏幕向下水平放在桌面上)。在这个旋转过程中，values[1]会在 0 到 -180 之间变化，也就是说，从手机顶部抬起时，values[1]的值会逐渐变小，直到等于 -180。如果从手机底部开始抬起，直到将手机沿 X 轴旋转 180°，这时 values[1]会在 0 到 180 之间变化。也就是 values[1]的值会逐渐增大，直到等于 180。可以利用 values[1]和下面要介绍的 values[2]来测量桌子等物体的倾斜度。

values[2]：表示手机沿着 Y 轴的滚动角度。取值范围是 $-90 \leqslant$ values[2] $\leqslant 90$。假设将手机屏幕朝上水平放在桌面上，这时如果桌面是平的，values[2]的值应为 0。将手机左侧逐渐抬起时，values[2]的值逐渐变小，直到手机垂直于桌面放置，这时 values[2]的值是 -90。将手机右侧逐渐抬起时，values[2]的值逐渐增大，直到手机垂直于桌面放置，这时 values[2]的值是 90。在垂直位置时继续向右或向左滚动，values[2]的值会继续在 -90 至 90 之间变化。

6.1.2 加速传感器

该传感器的 values 变量的 3 个元素值分别表示 X、Y、Z 轴的加速值。例如，水平放在桌面上的手机从左侧向右侧移动，values[0]为负值；从右侧向左侧移动，values[0]为正值。读者可以通过本节的例子来体会加速传感器中的值的变化。要想使用相应的传感器，仅实现 SensorEventListener 接口是不够的，还需要使用下面的代码来注册相应的传感器。

Java 代码：

```
    // 获得传感器管理器
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    // 注册方向传感器
sm.registerListener(this, sm.getDefaultSensor(Sensor.TYPE_ORIENTATION), SensorManager.
SENSOR_DELAY_FASTEST);
    // 获得传感器管理器
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    // 注册方向传感器
sm.registerListener(this, sm.getDefaultSensor(Sensor.TYPE_ORIENTATION), SensorManager.
SENSOR_DELAY_FASTEST);
```

如果想注册其他的传感器，可以改变 getDefaultSensor 方法的第一个参数值，例如，注册加速传感器可以使用 Sensor. TYPE_ACCELEROMETER。在 Sensor 类中还定义了很多传感器常量，但要根据手机中实际的硬件配置来注册传感器。如果手机中没有相应的传感器硬件，就算注册了相应的传感器也不起任何作用。getDefaultSensor 方法的第二个参数表示获得传感器数据的速度。SensorManager. SENSOR_DELAY_ FASTEST 表示尽可能快地获得传感器数据。除了该值以外，还可以设置 3 个获得传感器数据的速度值，这些值如下：

（1）SensorManager. SENSOR_DELAY_NORMAL：默认的获得传感器数据的速度。

（2）SensorManager. SENSOR_DELAY_GAME：如果利用传感器开发游戏，建议使用该值。

（3）SensorManager. SENSOR_DELAY_UI：如果使用传感器更新 UI 中的数据，建议使用该值。

6.1.3 重力传感器

加速度传感器的类型常量是 Sensor. TYPE_GRAVITY。重力传感器与加速度传感器使用同一套坐标系。values 数组中三个元素分别表示了 X、Y、Z 轴的重力大小。Android SDK 定义了一些常量，用于表示星系中行星、卫星和太阳表面的重力。下面就来温习一下天文知识，将来如果在地球以外用 Android 手机，也许会用得上。

```
public static final float GRAVITY_SUN = 275.0f;
public static final float GRAVITY_MERCURY = 3.70f;
public static final float GRAVITY_VENUS = 8.87f;
public static final float GRAVITY_EARTH = 9.80665f;
public static final float GRAVITY_MOON = 1.6f;
public static final float GRAVITY_MARS = 3.71f;
public static final float GRAVITY_JUPITER = 23.12f;
public static final float GRAVITY_SATURN = 8.96f;
public static final float GRAVITY_URANUS = 8.69f;
public static final float GRAVITY_NEPTUNE = 11.0f;
public static final float GRAVITY_PLUTO = 0.6f;
public static final float GRAVITY_DEATH_STAR_I = 0.000000353036145f;
public static final float GRAVITY_THE_ISLAND = 4.815162342f;
public static final float GRAVITY_SUN = 275.0f;
```

```
public static final float GRAVITY_MERCURY = 3.70f;
public static final float GRAVITY_VENUS = 8.87f;
public static final float GRAVITY_EARTH = 9.80665f;
public static final float GRAVITY_MOON = 1.6f;
public static final float GRAVITY_MARS = 3.71f;
public static final float GRAVITY_JUPITER = 23.12f;
public static final float GRAVITY_SATURN = 8.96f;
public static final float GRAVITY_URANUS = 8.69f;
public static final float GRAVITY_NEPTUNE = 11.0f;
public static final float GRAVITY_PLUTO = 0.6f;
public static final float GRAVITY_DEATH_STAR_I = 0.000000353036145f;
public static final float GRAVITY_THE_ISLAND = 4.815162342f;
```

6.1.4 光线传感器

光线传感器的类型常量是 Sensor. TYPE_LIGHT。values 数组只有第一个元素(values[0])有意义。表示光线的强度。最大的值是 120 000.0f。Android SDK 将光线强度分为不同的等级,每一个等级的最大值由一个常量表示,这些常量都定义在 SensorManager 类中,代码如下:

```
public static final float LIGHT_SUNLIGHT_MAX = 120000.0f;
public static final float LIGHT_SUNLIGHT = 110000.0f;
public static final float LIGHT_SHADE = 20000.0f;
public static final float LIGHT_OVERCAST = 10000.0f;
public static final float LIGHT_SUNRISE = 400.0f;
public static final float LIGHT_CLOUDY = 100.0f;
public static final float LIGHT_FULLMOON = 0.25f;
public static final float LIGHT_NO_MOON = 0.001f;
public static final float LIGHT_SUNLIGHT_MAX = 120000.0f;
public static final float LIGHT_SUNLIGHT = 110000.0f;
public static final float LIGHT_SHADE = 20000.0f;
public static final float LIGHT_OVERCAST = 10000.0f;
public static final float LIGHT_SUNRISE = 400.0f;
public static final float LIGHT_CLOUDY = 100.0f;
public static final float LIGHT_FULLMOON = 0.25f;
public static final float LIGHT_NO_MOON = 0.001f;
```

上面的常量只是临界值。读者在实际使用光线传感器时要根据实际情况确定一个范围。例如,当太阳逐渐升起时,values[0]的值很可能会超过 LIGHT_SUNRISE,当 values[0]的值逐渐增大时,就会逐渐越过 LIGHT_OVERCAST,而达到 LIGHT_SHADE,当然,如果天气特别好的话,也可能会达到 LIGHT_SUNLIGHT,甚至更高。

6.1.5 陀螺仪传感器

陀螺仪传感器的类型常量是 Sensor. TYPE_GYROSCOPE。values 数组的三个元素表示的含义如下:

values[0]: 绕 *X* 轴旋转的角速度。

values[1]：绕 Y 轴旋转的角速度。

values[2]：绕 Z 轴旋转的角速度。

当手机逆时针旋转时，角速度为正值，顺时针旋转时，角速度为负值。陀螺仪传感器经常被用来计算手机已转动的角度，代码如下：

```
private static final float NS2S = 1.0f / 1000000000.0f;
private float timestamp;
public void onSensorChanged(SensorEvent event)
    {
        if (timestamp != 0)
        {
        // event.timesamp 表示当前的时间,单位是纳秒(百万分之一毫秒)
                final float dT = (event.timestamp - timestamp) * NS2S;
                angle[0] += event.values[0] * dT;
                angle[1] += event.values[1] * dT;
                angle[2] += event.values[2] * dT;
        }
        timestamp = event.timestamp;
    }
private float timestamp;
public void onSensorChanged(SensorEvent event) {
if (timestamp != 0)
{    // event.timesamp 表示当前的时间,单位是纳秒(百万分之一毫秒)
final float dT = (event.timestamp - timestamp) * NS2S;
angle[0] += event.values[0] * dT;
angle[1] += event.values[1] * dT;
angle[2] += event.values[2] * dT; }
timestamp = event.timestamp;
}
```

上面代码中通过陀螺仪传感器相邻两次获得数据的时间差(dT)来分别计算在这段时间内手机绕 X、Y、Z 轴旋转的角度，并将值分别累加到 angle 数组的不同元素上。

6.1.6 其他传感器

在前面几节介绍了加速度传感器、重力传感器、光线传感器、陀螺仪传感器及方向传感器。除了这些传感器外，Android SDK 还支持如下的几种传感器：

- 近程传感器(Sensor. TYPE_PROXIMITY)
- 线性加速度传感器(Sensor. TYPE_LINEAR_ACCELERATION)
- 旋转向量传感器(Sensor. TYPE_ROTATION_VECTOR)
- 磁场传感器(Sensor. TYPE_MAGNETIC_FIELD)
- 压力传感器(Sensor. TYPE_PRESSURE)
- 温度传感器(Sensor. TYPE_TEMPERATURE)

关于这些传感器的使用方法及与这些传感器相关的常量、方法，读者可以参阅官方文档。

虽然 Android SDK 定义了十多种传感器，但并不是每一部手机都完全支持这些传感

器。例如，Google Nexus S 支持其中的 9 种传感器(不支持压力和温度传感器)，而 HTC G7 只支持其中的 5 种传感器。如果使用了手机不支持的传感器，一般不会抛出异常，但也无法获得传感器传回的数据。读者在使用传感器时最好先判断当前的手机是否支持所使用的传感器。

6.1.7 测试手机中有哪些传感器

我们可以通过如下三步使用传感器：

(1) 编写一个截获传感器事件的类。该类必须实现 android. hardware. SensorEventListener 接口。

(2) 获得传感器管理对象(SensorManager 对象)。

(3) 使用 SensorManager. registerListener 方法注册指定的传感器。

通过上面三步已经搭建了传感器应用程序的框架。而具体的工作需要在 SensorEventListener 接口的 onSensorChanged 和 onAccuracyChanged 方法中完成。SensorEventListener 接口的定义如下：

```
package android. hardware;
public interface SensorEventListener
{
//传感器数据变化时调用
public void onSensorChanged(SensorEventevent);
//传感器精确度变化时调用
public void onAccuracyChanged(Sensorsensor, int accuracy);
}
public interfaceSensorEventListener
{ //传感器数据变化时调用
public void onSensorChanged(SensorEventevent);
//传感器精确度变化时调用
public void onAccuracyChanged(Sensorsensor, int accuracy); }
```

SensorManager 对象通过 getSystemService 方法获得，代码如下：

```
SensorManagersensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
```

通常手机中包含了若干个传感器模块(如方向传感器、光线传感器等)，因此，注册传感器需要指定传感器的类型，如下面的代码注册了光线传感器：

```
sensorManager. registerListener(this, sensorManager. getDefaultSensor(Sensor. TYPE_LIGHT),
SensorManager. SENSOR_DELAY_FASTEST);
sensorManager. registerListener ( this, sensorManager. getDefaultSensor ( Sensor. TYPE _ LIGHT),
SensorManager. SENSOR_DELAY_FASTEST);
```

registerListener 方法有三个参数。第一个参数是实现 SensorEventListener 接口的对象。第二个参数用于指定传感器的类型。Android SDK 预先定义了表示各种传感器的常量，这些常量都被放在 Sensor 类中。例如，上面代码中的 Sensor. TYPE_LIGHT。第三个

参数表示传感器获得数据的速度。该参数可设置的常量如下：

SENSOR_DELAY_FASTEST：以最快的速度获得传感器数据。

SENSOR_DELAY_GAME：适合于在游戏中获得传感器数据。

SENSOR_DELAY_UI：适合于在 UI 控件中获得传感器数据。

SENSOR_DELAY_NORMAL：以一般的速度获得传感器的数据。

上面四种类型获得传感器数据的速度依次递减。从理论上说，获得传感器数据的速度越快，消耗的系统资源越大。因此建议读者根据实际情况选择适当的速度获得传感器的数据。

如果想停止获得传感器数据，可以使用 unregisterSensor 方法注销传感器事件对象。unregisterSensor 方法的定义如下：

```
public voidunregisterListener(SensorEventListener listener)
public voidunregisterListener(SensorEventListener listener, Sensor sensor)
```

unregisterSensor 方法有两个重载形式。第一个重载形式用于注销所有的传感器对象。第二个重载形式用于注销指定传感器的事件对象。其中 Sensor 对象通过 SensorManager. getDefaultSensor 方法获得。getDefaultSensor 方法只有一个 int 类型的参数，表示传感器的类型。如 Sensor. TYPE_LIGHT 表示光线传感器。

注意：一个传感器对象可以处理多个传感器。也就是说，一个实现 SensorEventListener 接口的类可以接收多个传感器传回的数据。为了区分不同的传感器，需要使用 Sensor. getType 方法来获得传感器的类型。getType 方法的应用将在本节的例子中详细介绍。

通过 SensorManager. getSensorList 方法可以获得指定传感器的信息，也可以获得手机支持的所有传感器的信息，代码如下：

```
//获得光线传感器
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_LIGHT);
//获得手机支持的所有传感器
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

6.2 系统总体介绍

6.2.1 WSN 概述

无线传感器网络(Wireless Sensor Network，WSN)系统是当前在国际上备受关注的、涉及多学科高度交叉、知识高度集成的前沿热点研究领域。它综合了传感器技术、嵌入式计算技术、现代网络及无线通信技术、分布式信息处理技术等。无线传感器网络就是由部署在检测区域内大量的廉价卫星传感器节点组成，通过无线通信方式形成的一个多跳的自组织的网络系统，目的是协作地感知、采集和处理网络覆盖区域中感知的对象信息，并发送给观察者。

构成 WSN 的三要素是传感器、感知对象、观察者。

6.2.2 历史及发展现状

1. 国外

WSN 起源于美国，根源可追溯到 1978 年由国防部高级研究计划署（DARPA）在卡内基-梅隆大学发起的分布式传感器研讨会。具有代表性的项目包括：1993—1999 年间由美国国防部高级研究计划署（DARPA）资助，加州大学洛杉矶分校（UCLA）承担的 WINS 项目；1999—2001 年间由 DAPRA 资助 UC Berkeley 承担的 Smart Dust 项目。1998—2002 年 DARPA 资助，加州大学伯克利分校等 25 个机构联合承担的 SensIT 计划；1999—2004 年间海军研究办公室的 SeaWeb 计划等。

2. 国内

中国的一些研究机构近年开始研究：中国科学技术大学、清华大学、中科院计算所、上海微系统所、沈阳自动化所及合肥智能所等研究单位。之所以国内外都投入巨资研究机构纷纷开展无线传感器网络的研究，很大程度归功于其广阔的应用前景和对社会生活的巨大影响。

6.2.3 WSN 的应用

无线传感器网络具有可快速部署、可自组织、隐蔽性强和高容错性的特点，因此非常适合在军事上应用。利用无线传感器网络能够实现对敌军兵力和装备的监控、战场的实时监视、目标的定位、战场评估、核攻击和生物化学攻击的监测和搜索等功能。目前国际许多机构的课题都是以战场需求为背景展开的。例如，美军开展的如 C4KISR 计划、灵巧传感器网络通信、无人值守地面传感器群、传感器组网系统、网状传感器系统 CEC，等等。

1. 军事领域

在军事领域应用方面，该项技术的远景目标是：利用飞机或火炮等发射装置，将大量廉价传感器节点按照一定的密度布放在待测区域内，对周边的各种参数，如温度、湿度、声音、磁场、红外线等各种信息进行采集，然后由传感器自身构建的网络，通过网关、互联网、卫星等信道，传回信息中心。

2. 民用领域

根据需要，人们可以在待测区域安放不同功能的传感器并组成网络，长期大面积地监测微小的气候变化，包括温度、湿度、风力、大气、降雨量，收集有关土地的湿度、氮浓缩量和土壤 pH 值等，从而进行科学预测，帮助农民抗灾、减灾，科学种植，获得较高的农作物产量。

2002 年，英特尔公司率先在俄勒冈建立了世界上第一个无线葡萄园。传感器节点被分布在葡萄园的每个角落，每隔一分钟检测一次土壤温度、湿度或该区域有害物的数量，以确保葡萄可以健康生长。研究人员发现，葡萄园气候的细微变化可极大地影响葡萄酒的质量。通过长年的数据记录及相关分析，便能精确地掌握葡萄酒的质地与葡萄生长过程中的日照、温度、湿度的确切关系。这是一个典型的精准农业、智能耕种的实例。

无线传感器网络可以广泛地应用于生态环境监测、生物种群研究、气象和地理研究、洪水和火灾检测。以下列出一些常见的应用领域：

- 可通过跟踪珍稀鸟类、动物和昆虫的栖息、觅食习惯等，进行濒临种群的研究。
- 可在河流沿线布设传感器节点，随时监测水位及相关水资源被污染的信息。

- 在山区中泥石流、滑坡等自然灾害容易发生的地方布设节点，可提前发出预警，以便做好准备，采取相应措施，防止进一步的恶性事故的发生。
- 可在重点保护林区铺设大量节点，随时监控内部火险情况，一旦有危险，可立刻发出警报，并给出具体方位及当前火势大小。
- 布放在地震、水灾、强热带风暴灾害地区、边远或偏僻野外地区，用于紧急和临时场合应急通信。

各类大型工程的安全施工及监控是建筑设计单位长期关注的问题。如已经建成的三峡工程，正在建设的苏通大桥、渤海海域的大量的海洋平台和海底管线、2008 年的奥运场馆等。采用无线传感器网络，可以让大楼、桥梁和其他建筑物能够自身感觉并意识到它们的状况，使得安装了传感器网络的智能建筑自动告诉管理部门它们的状态信息，从而可以让管理部门按照优先级进行定期的维修工作。

对珍贵的古老建筑进行保护，是文物保护单位长期以来的一个工作重点。将具有温度、湿度、压力、加速度、光照等传感器的节点布放在重点保护对象中，无须拉线钻孔，便可有效地对建筑物进行长期的监测。

此外，对于珍贵文物而言，在保存地点的墙角、天花板等位置布设传感节点，监测环境的温度、湿度是否超过安全值，可以更妥善地保护展览品的品质。

6.2.4 WSN 的体系结构

1. 传感器网络结构

传感器网络结构如图 6-1 所示。

2. 传感器节点结构

传感器节点结构如图 6-2 所示。

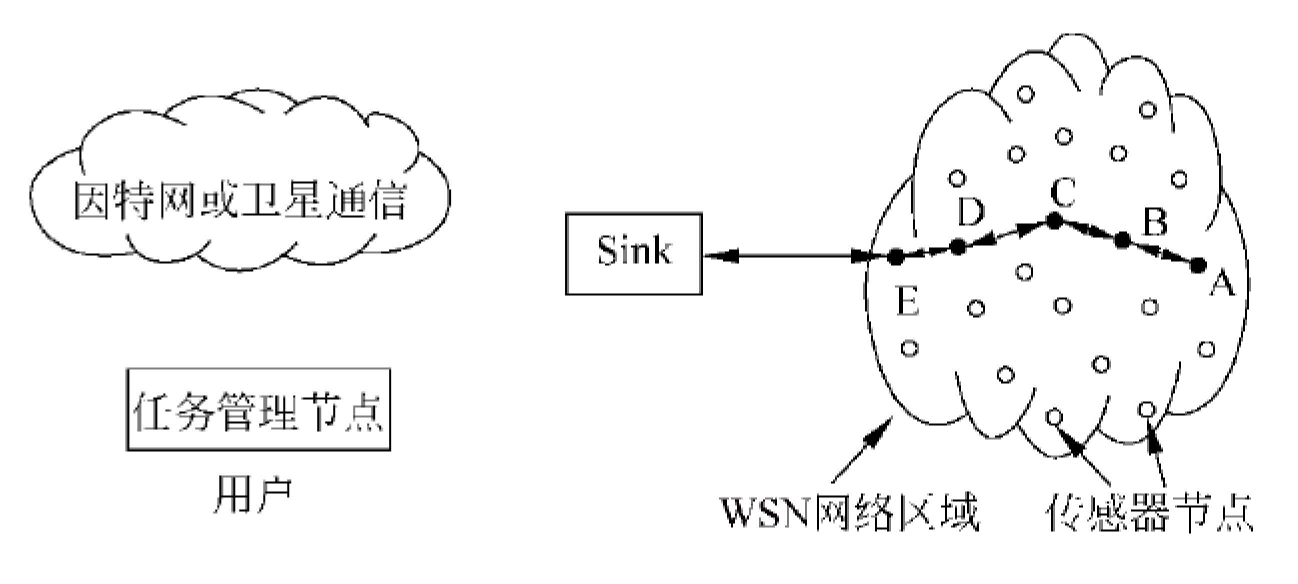


图 6-1 传感器网络结构

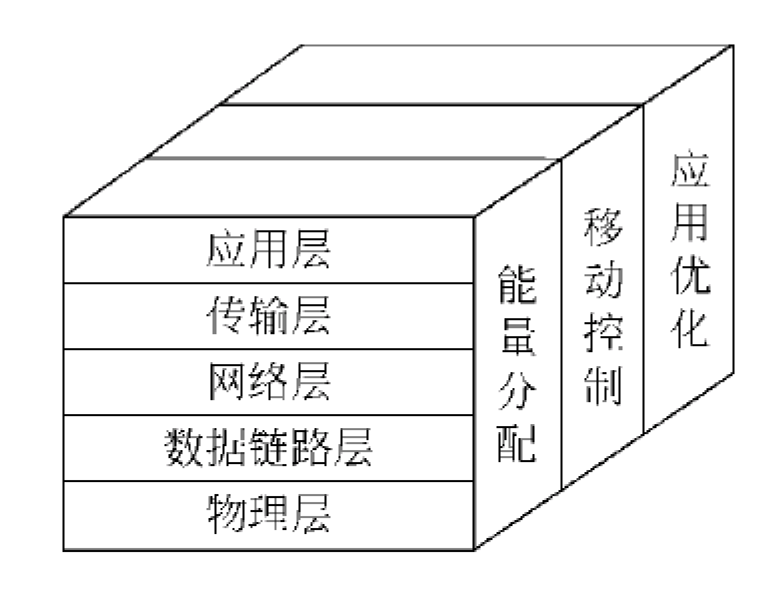


图 6-2 传感器节点结构

3. 传感器网络协议栈

传感器网络协议栈结构如图 6-3 所示。

6.2.5 WSN 的特征

WSN 有如下特征：

(1) 硬件资源有限。节点由于受价格、体积和功耗的限制，其计算能力、程序空间和内存空间比普通的计算机功能要弱很多。这一点决定了在节点操作系统设计中，协议层次不能太复杂。

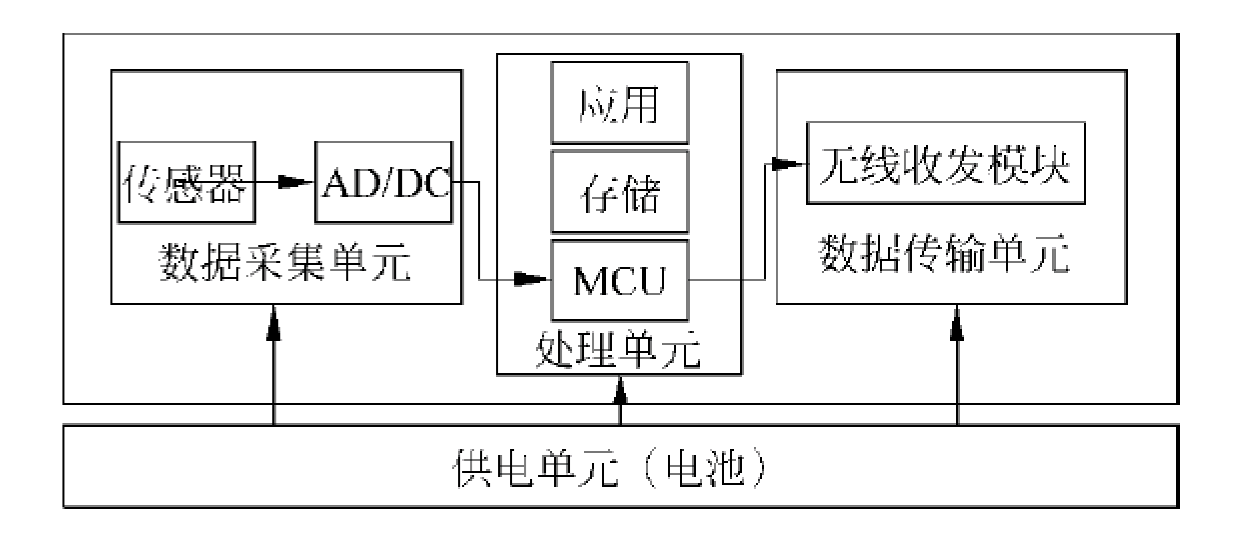


图 6-3　传感器网络协议栈结构

（2）电源容量有限。网络节点由电池供电，电池的容量一般不是很大。其特殊的应用领域决定了在使用过程中，不能给电池充电或更换电池，一旦电池能量用完，这个节点也就失去了作用（死亡）。因此在传感器网络设计过程中，任何技术和协议的使用都要以节能为前提。

（3）无中心。无线传感器网络中没有严格的控制中心，所有节点地位平等，是一个对等式网络。节点可以随时加入或离开网络，任何节点的故障不会影响整个网络的运行，具有较强的抗毁性。

（4）自组织。网络的布设和展开无须依赖于任何预设的网络设施，节点通过分层协议和分布式算法协调各自的行为，节点开机后就可以快速、自动地组成一个独立的网络。

（5）多跳路由。网络中节点通信距离有限，一般在几十到几百米范围内，节点只能与它的邻居直接通信。如果希望与其射频覆盖范围之外的节点进行通信，则需要通过中间节点进行路由。固定网络的多跳路由使用网关和路由器来实现，而无线传感器网络中的多跳路由是由普通网络节点完成的，没有专门的路由设备。这样每个节点既可以是信息的发起者，也是信息的转发者。

（6）动态拓扑。无线传感器网络是一个动态的网络，节点可以随处移动；一个节点可能会因为电池能量耗尽或其他故障，退出网络运行；一个节点也可能由于工作的需要而被添加到网络中。这些都会使网络的拓扑结构随时发生变化，因此网络应该具有动态拓扑组织功能。

（7）节点数量众多，分布密集。为了对一个区域执行监测任务，往往有成千上万传感器节点投放到该区域。传感器节点分布非常密集，利用节点之间高度连接性来保证系统的容错性和抗毁性。

无线传感器网络是一个典型的分布式系统，正是由于其以上特点，使得无线传感器网络的设计过程中要尽量地使用分布式算法，并且对于一个分布式系统需要考虑的问题，比如容错、安全、可靠通信等都需要考虑。

6.2.6　WSN 未来发展前景

20 世纪 90 年代初，美国的 Weiser 博士提出“普适计算”这一新概念，开始了人类向未来计算机时代探索的进程。所谓普适计算指的是，在普适环境下使人们能够使用任意设备、通过任意网络、在任意时间都可以获得一定质量的网络服务的技术。普适计算被认为是一种能包含各种设备（计算机、汽车、娱乐设备和信息设备等）模式，无论何时何地，只要需要，就可以通过某种设备访问到所需的信息。

物联网在国际上又称为传感网，据悉，这是继计算机、Internet与移动通信网之后的又一次信息产业浪潮。南京邮电大学校长、博士生导师杨震教授说，世界上的万事万物，小到手表、钥匙，大到汽车、楼房，只要嵌入一个微型感应芯片，把它变得智能化，这个物体就可以"自动开口说话"。再借助无线网络技术，人们就可以和物体"对话"，物体和物体之间也能"交流"，这就是物联网。"如果物联网再搭上Internet这个桥梁，在世界任何一个地方我们都可以即时获取万事万物的信息。可以这么说，物联网加上Internet等于智慧地球。"物联网用途广泛，可运用于城市公共安全、工业安全生产、环境监控、智能交通、智能家居、公共卫生、健康监测等多个领域，让人们享受到更加安全轻松的生活。

举几个例子来说，从成都开车到重庆，上车后，只要设置好目的地便可随意睡觉、看电影，车载系统会通过从路面接收到的信号智能行驶；不住在医院，只要通过一个小小的仪器，医生就能24小时监控病人的体温、血压、脉搏；下班了，只要用手机发出一个指令，家里的电饭煲就会自动加热做饭，空调开始降温……

通过"物联网"的逐步实现和提升，每个人的生活都将向此靠拢。物联网将把各种信息传感设备与Internet结合起来而形成的一个巨大网络。简单一点说，如果物联网顺利普及，就意味着几乎所有的电器、家居用品、汽车制造都急需更新换代。

具体来说，就是通过安装信息传感设备，如射频识别(RFID)装置、红外感应器、全球定位系统、激光扫描器等，将所有的物品都与网络连接在一起，方便识别和管理。电视、洗衣机、空调甚至自行车、门锁和血压计上都能使用。

专家预测10年内，物联网就可能大规模普及，将广泛运用于智能交通、环境保护、政府工作、公共安全、平安家居、智能消防、工业监测、老人护理、个人健康等多个领域，一个上万亿元规模的高科技市场就此诞生。

有人预测，如果物联网全部构成，其产业要比Internet大30倍！物联网将会成为下一个万亿元级的通信业务。

6.3 系统模块介绍

6.3.1 无线传感器节点网络

在节点的功能设计和实现方面，目前常用的节点均为采用分立元器件的系统集成技术。已出现的多种节点的设计和平台套件，在体系结构上有相似性，主要区别在于采用了不同的微处理器，如AVR系列和MSP430系列等；或者采用了不同的射频芯片或通信协议，比如采用自定义协议、802.11协议、ZigBee协议、蓝牙协议及UWB通信方式等。典型的节点包括Berkeley Motes，Sensoria WINS，MIT μAMPs，Intel iMote，Intel XScale nodes，CSRIO研究室的CSRIO节点、Tmote、ShockFish公司的TinyNode、耶鲁大学的XYZ节点、smart-its BTNodes等。国内也出现诸多研究开发平台套件，包括中科院计算所的EASI系列，中科院软件所、清华大学、中科大、哈工大、大连海事大学等单位也都已经开发出了节点平台支持网络研究和应用开发。

这些由不同公司及研究机构研制的无线节点在硬件结构上基本相同，包括处理器单元、存储器单元、射频单元、扩展接口单元、传感器及电源模块。其中，核心部分为处理器模块及

射频通信模块。处理器决定了节点的数据处理能力和运行速度等，射频通信模块决定了节点的工作频率和无线传输距离，它们的选型能在很大程度上影响节点的功能、整体能耗和工作寿命。

目前问世的传感节点（负责通过传感器采集数据的节点）大多使用如下几种处理器：ATMEL 公司 AVR 系列的 ATMega128L 处理器、TI 公司生产的 MSP430 系列处理器，而汇聚节点（负责会聚数据的节点）则采用了功能强大的 ARM 处理器、8051 内核处理器、ML67Q500x 系列或 PXA270 处理器。这些处理器的性能综合比较见表 6-1。

表 6-1　无线传感器网络节点中采用的处理器性能比较

性能参数\处理器	ATMega128L	MSP430F1611	ML67Q5002	PXA270
总线带宽(位)	8	16	32	32
时钟频率(MHz)	7.3728	4	60	≤520
工作电压(V)	3.3	3.3	3.3	2.5/3.3
工作电流	20mA	600μA	120mA	—
休眠电流	25μA	4.3μA	20μA	—
内部 FLASH	128KB	48KB+256B	256KB	—
内部 SRAM	4KB	10KB	32KB	—

在无线传感器网络中，广泛应用的底层通信方式包括使用 ISM 波段的普通射频通信、具有 802.15.4 协议和蓝牙通信协议的射频通信。使用普通 ISM 频段的无线传感器网络节点根据在不同的国家和地区对于 ISM 波段频率的定义不同，一般将通信频率设置为 433MHz 或者 868/915MHz。在硬件的设计中，所采用的芯片包括 Chipcon 公司的 CC1000，Nordic 公司的 nrf903，Semtech 公司的 XE1205。还有部分无线传感器网络节点使用了带有 802.15.4/ZigBee 协议的通信芯片，具有这样协议的芯片包括 Chipcon 公司的 CC2420、RFWave 公司的 RFW102 芯片组。还有部分节点采用了 Bluetooth 协议进行通信，具有 Bluetooth 协议的芯片组包括 Ericsson 公司生产的 ROK 101 007 等。

上述这些射频芯片的性能比较以及代表性节点的性能比较见表 6-2 和表 6-3。

表 6-2　无线传感器网络节点中采用的射频模块综合比较

性能参数\射频模块	CC1000	nrf903	XE1205	CC2420	RFW102	ROK101007
通信频率(MHz)	300～1000	433/868/915	433/868/915	2400	2400	2400
调制方式	FSK	GFSK	FSK	O-QPSK	ASK	GFSK
编码方式	NRZ/曼彻斯特	直接同/异步	NRZ	DSSS	DSSS	FHSS
工作电压(V)	2.7～5	2.7～3.3	2.4～3.9	2.1～3.6	2.4～3.6	3～5
最大输出功率(dbm)	10	10	15	0	7	4
灵敏度(dbm)	−110	−104	−110	−94	−77	−40
传输速度(kbps)	76.8	76.8	153.2	250	1000	1000
接收电流(mA)	7.4	18.5	14	19.7	7	
协议	无	无	无	802.15.4/ZigBee	802.15.4/ZigBee	Bluetooth

表 6-3　无线传感器网络节点综合比较

无线节点\参数	处理器	射频芯片	扩展存储器	特　　点
Mica2	ATMega128L	CC1000	512KB	低功耗、电源监测、全球唯一地址
Mica z	ATMega128L	CC2420	512KB	低功耗
Tmote	MSP430F1611	CC2420	—	USB 接口、超低功耗,集成温湿度、光照度传感器
CSIRO	ATMega128L	nrf903	4KB	GFSK 调制,误码率低
TinyNode 584	MSP430F1611	XE1205	512KB	无线传输速率高,距离远
Imote2	PXA270	CC2420	32MB	USB 接口,处理功能强大,可以进行音视频处理
XYZ	ML67Q5002	CC2420	—	处理功能强大
BTNodes	ATMega103	ROK101 007	64KB	低功耗,Bluetooth 通信
EASI210	ATMega128L	CC1000	512KB	电源监测、低功耗,高稳定性,全球唯一地址

由表 6-3 可以看出,各公司生产的不同无线传感器网络节点根据所选用的核心处理器与射频通信芯片及扩展功能的不同,分别具有不同的特点。采用 MSP430 单片机具有的超低功耗特点,如 Tmote;采用了超强处理器的节点更加擅长处理大数据量,适用于高速通信、环境复杂、需要强大数据处理能力的场合,如 imote 2 及 XYZ 节点;使用 ATMega128L 芯片处理器则在性能和功耗之间较为平衡,处理速度较快,而功耗又相对较低,是一种折中的方案。在射频方面,采用 2.4GHz 无线通信频率的节点包括使用了 802.15.4/ZigBee 通信协议及 Bluetooth 通信协议,这两种方式将 MAC 层以下的通信协议固化在模块中,不需要进一步进行开发,步骤简化,更具兼容性,如 Mica z、Tmote、Imote2 及 XYZ 及 BTNodes 节点,采用其他射频芯片的节点由于其通信频率比较低,因此在通信距离上较有优势,还可开发满足需要的 MAC 协议。

尽管已经出现了以上诸多类型的节点,但这些节点基本上还都是实验系统,是支持研究和二次开发的平台,尚没有实现系列化和标准化的工业级设计,距离真正的实际应用需要,在技术成熟度上和功能上都尚有很大的差距,成本也比较高。

支持系统异构性的节点目前为数不多,CrossBow 公司生产的 SPB400 stargate 网关节点使用了 PXA255 处理器,操作系统采用了 Linux;而传感节点则采用 mica 系列,使用 Tiny OS 操作系统。该网关节点具有强大的数据处理功能,并有多种接口,包括串行口、USB、以太网及 JTAG 接口等,和 mica 节点插接使实现射频通信。目前为了支持异构网络(包括网络中采用不同的或混合的无线通信方式)而需要的具有更强系统异构性的节点不多见,Intel XScale 是一个例子,在使用了 PXA250 XScale 处理器的网关上增加 802.11 通信方式,使得网关节点间具有较强的通信能力,网络中其他传感节点使用非 802.11 协议(如 802.15.4/ZigBee)的方式进行无线通信,以支持分层的异构网络应用。目前,对异构网络的研究大多数是针对异构网络通信协议以及算法,以及 Mesh 网络的体系结构等,尚缺乏足够多样化的实际节点系统平台作为支持。因此,支持系统异构性的、系列化的无线传感器网络节点正是当前急需启动的研究内容。

集成片上系统是向下一代节点发展的必然趋势,它在物理设计上进行改进来减少节点的体积、成本和功耗,是从根本上解决低成本和高可靠性的技术手段。下一代节点的典型代

表有 U.C. Berkeley 的 Smart Dust 以及 PicoRadio，CSEM 的 WiseNET，芬兰坦佩雷技术大学的 Multi-Radio WSN Platform 等，它们均采用了 SOC 技术，在一个芯片上集成了 CPU、自定义逻辑模块、甚至射频模块和传感器模块，用这样的芯片辅以较少的外设来实现传感器节点。目前此类节点的开发，一般先在 FPGA 开发平台上进行，验证完成后再转为 ASIC 量产。由于能够自行选择和设计逻辑模块，此类平台的开发灵活性有了很大的提高，在 FPGA 验证完成的情况下，配上先进的工艺来设计 ASIC 芯片，可以大幅度地减少节点的功耗、体积和成本并且提高可靠性。不过此类节点的开发比较复杂，目前多为各个实验室自行开发各自的平台。不过随着 SOC 技术的发展和 IP(Intellectual Property)模块的普及，此类节点的开发会越来越容易。

Smart Dust 是 1999 年 U.C. Berkeley 在美国国防部委托下开发出的一套无线传感器网络节点，采用光通信方式。同时，它采用了 MEMS 技术，融合了硅微加工、光刻铸造成型(LIGA)和精密机械加工等多种微加工技术，使得它的长度在 5mm 之内。Smart Dust 采用了 SOC 的方式，在一个芯片中集成了传感器、处理器、光通信装置等器件，成功地达到了减小体积，降低功耗的目的。

PicoRadio 研究组属于 Berkeley 的无线研究中心。为了研发采用 SOC 技术的无线传感器网络节点 PicoNode，2002 年它们设计了 PicoRadio Test Bed 这一研发平台，它由处理器板、电源板、通信板和传感器板四个板块叠加而成。其中处理器板采用了 ARM 1100 的 CPU 和 Xilinx XC4020XLA 的 FPGA 作为处理器，通信板采用蓝牙作为通信方式。在开发中，应用层和高层次的网络协议用软件的方式通过 CPU 来实现，而低层次的网络协议以及蓝牙芯片的控制则通过硬件编程的方式用 FPGA 来实现。由于 PicoRadio Test Bed 只是一个测试平台，还没有实现真正意义上的 SoC，因此 PicoRadio Test Bed 体积和功耗还难以让人满意。其研究还表明，在运行同样 MAC 协议的相同工艺下不同平台在功耗方面有较大差异，以 ASIC 为最低。因此，只要将 PicoRadio Test Bed 转化为 ASIC 芯片，则它的功耗和体积都可望大大下降。

WiseNET 是瑞士 CMES 开发的一套无线传感器网络节点芯片，WiseNET 采用 SOC 技术，专门为无线传感器网络而设计。在一块芯片上集成了射频模块、MAC 协议、采用 Cool-RISC 结构的微控制器、电源模块、ADC 模块以及 SPI、I2C 的接口，用户只需外接电池、传感器和天线即可将它制作成节点。从功能、体积和功耗上它都比用通用的 CPU 设计出的传感器节点有较大的改进。

如果说前三种节点体现的是 SOC 节点在体积和功耗上的优势，坦佩雷技术大学 芬兰(Tampere University of Technology)的 Multi-Radio WSN Platform 则体现出了 SOC 节点在硬件灵活性上的优势。与往常的节点不同，Multi-Radio WSN Platform 采用了四个射频模块，用频分的方式在 4 个频段上同时进行数据收发，可达到较高的数据传输速率。采用的是 Altera Cyclone EP1C20 的 FPGA，并使用了 Nios II CPU 软核作为片上的处理器，同时它在 FPGA 上实现了四个射频芯片的接口模块，比建立一个射频控制模块来协调四个射频芯片的工作。

目前在国内开展面向下一代网络节点 SOC 的工作有中科院计算所的 EASISOC，并已经完成了一款具有简单功能的节点 FPGA 验证，目前正在开展高端 SOC 节点的设计验证工作。

6.3.2 采集终端

采集终端有两种：一种是 PC 上 AtosWin，安装于 Windows 上的采集程序；另一种是 AtosCE，安装于 ASK270(PXA270)上，基于 Microsoft WinCE 的采集程序。采集终端程序通过串口、USB 口、获取在网关板上的基站节点所汇聚的 RF 射频数据，然后在本地进行临时存储，同时可以方便地查看。

在用户指定操作或规定特定时长策略下，采集终端可以将数据上传至 Web 数据库服务器。

上传数据时，使用了 Web Services 技术，服务器无须配置 FTP 服务器、编写网页脚本，即可将新数据发送给服务器。

每个采集终端在服务器端看来都可以称之为区域，服务器负责存储全部区域的数据，用户通过服务器即可获取所有区域信息。

6.3.3 服务器

AtosServer 是运行于服务器上的服务程序集合，负责接收所有采集终端(区域)所采集的数据，并且按照区域代码，将所有数据存入数据库中，然后对外提供一个 Web 服务器功能。系统采用 B/S 架构，用户通过特定的终端即可随时随地访问服务器上的数据。

AtosServer 包括几大模块：

(1) AtosService：采集终端数据守护模块：使用.NET WebServices 技术，负责与采集终端进行交互，接收采集终端采集的数据。同时使用 ADO.NET 将接收到的数据存入对应的数据库中(SQL SERVER)。

(2) AtosBrowser：PC 终端服务程序，可用于访问服务器，随时随地获取各区域最新的数据，尽快掌握 WSN 数据动态。

(3) AtosMobile：移动终端的服务程序，由 J2ME 编写，可运行在大量的手机平台上，更为方便地利用手机移动性，可及时地获取服务器中的最新数据。

6.3.4 PC 终端 AtosBrowser

AtosBrowser 终端，可运行于 IE、FIREFOX 等目前流行的各类 WEB 浏览器，用户可以在任意一台安装有 WEB 浏览器的 PC 终端上访问服务器数据，掌控 WSN 网络。

PC 终端上采用了 FLASH、AJAX(WEB2.0)技术，实现了富客户端，用户可以直观地获取所需数据。并且可以将数据以各种方式方便地查看(实时地图系统、实时表单系统、动态图表系统等)，同时可以方便地在线统计、打印。可以对传感器网络获取环境数据的即时反应，迅速做出决策。PC 终端具有如下功能：

(1) 实时数据地图功能。用户选择的区域，加载对应区域的地图，同时加载该区域内的节点。用户可以手动更新地图数据或者设置一定时长的自动刷新。获取新数据后，用户可以选择对应的节点进行详细信息查看。同时可以切换到历史记录、图表等功能。

(2) 历史数据列表。对于特定的节点，用户查看其规定时间内的数据列表，可以选择将数据导出成 EXCEL 或者在线打印等。

(3) 图表功能。对应特定的节点，用户还可以使用其历史数据进行图表绘制，用 Flash

绘制出直观的图表,方便用户理解和决策。同时可以导出保存图片。

(4) 数据管理功能。通过特定的权限机制,可以允许管理员对数据库中已有的数据进行管理,维护数据库。

6.3.5 移动终端 AtosMobile

AtosMobile 软件采用 J2ME 编写,可以运行于手机、PDA 等移动设备上。受益于 Java 的跨平台性,系统几乎可以运行于所有的智能手机平台上。

AtosMobile 为用户提供了友好形象的界面,通过 GPRS 上网,可以从服务器上获取实时的 WSN 网络数据,同时在本地进行显示和操作。AtosMobile 提供功能如下:

(1) 实时数据地图功能。可以根据用户选择的区域,加载对应区域的地图,同时加载该区域内的节点。用户可以手动更新地图数据或者设置一定时长的自动刷新。获取新数据后,用户可以选择对应的节点进行详细信息查看。

(2) 历史数据列表。对于特定的节点,用户查看其规定时间内的数据列表。

(3) 图表功能。对应特定的节点,用户还可以使用其历史数据进行图表绘制,在手机上绘制出直观的图表,方便用户理解和决策。

同时提供用户开发的接口,用户可以方便地在本系统架构上开发自己的程序,以适应特定的应用需求。

6.3.6 协议分析助手 AtosAgent

OSAgent 是一个真实硬件调试分析框架,可以帮助研究者方便地记录网络数据、操控各节点进行实验、收集数据、统计分析数据。AtosAgent 具有如下特点:

(1) 完全兼容亿道的 WSN 开发套件。

(2) OSAgent 完全独立于用户程序之外。

(3) AtosAgent 主控程序掌控所有节点,方便地运行、收集、分析数据,其易用性大大降低硬件实验的门槛。

(4) 利用 OSAgent 框架,可以在真实环境中完整地调试自己的网络协议,测试性能。

6.4 Android 在 WSN 中的应用现状和前景

6.4.1 Android 在 WSN 中的应用现状

Android 系统的一大亮点就是对传感器的应用,利用传感器可以开发出很多新奇有趣的程序,小到水平仪、计数器,大到传感器游戏,本节主要介绍 Android 在 WSN 中的应用现状和前景。WSN 一般模型如图 6-4 所示。

WSN 节点模型如图 6-5 所示。

一个传感节点获得传感信息后,该信息以分布式或集中式的方式传输给网络中的其他节点,智能化网络由很多可以相互通信及进行数据处理的感应节点和行动节点组成,构成“智能环境”。

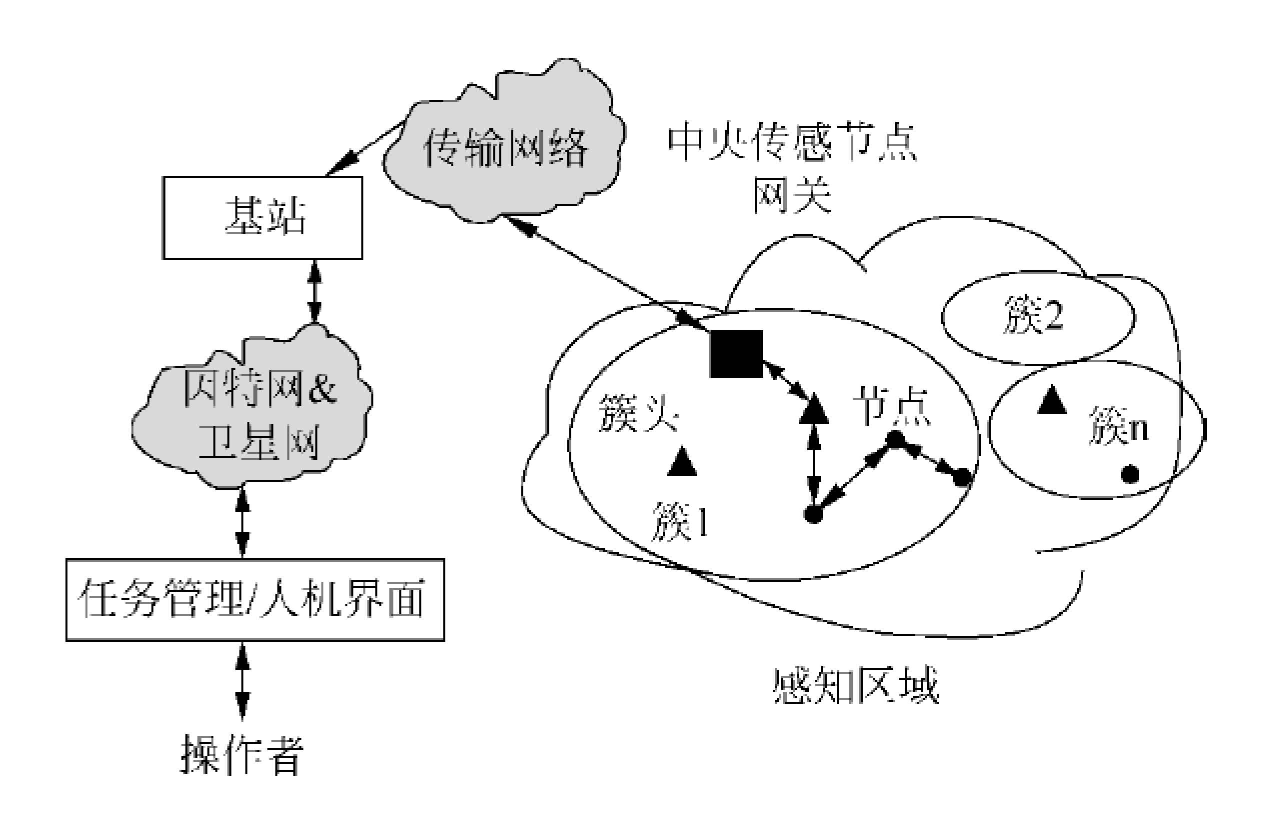


图 6-4　WSN 一般模型

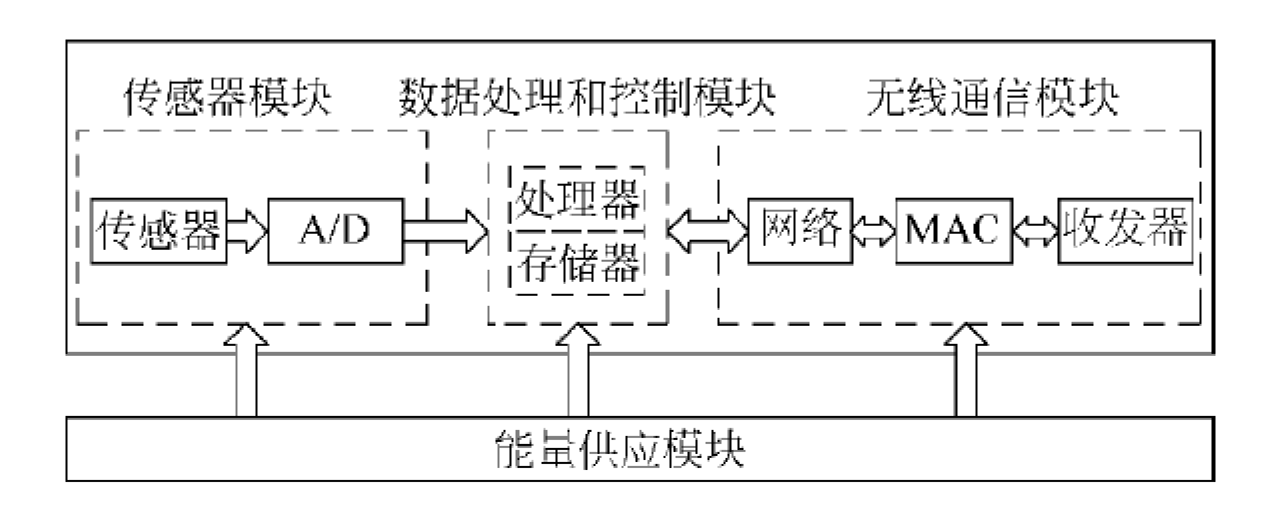


图 6-5　WSN 节点模型

传感器就如同神经末梢，系统就是其灵魂。传感器采集外界参数，系统来处理这些参数以确定接下来要做什么，最后由自组的无线网络发射出去。

目前 Android 系统下的传感器技术是非常多的，从 Android 1.5 开始，系统内置了对多达八种传感器的支持，比如：加速度传感器、陀螺仪、环境光照传感器、磁力传感器、方向传感器、压力传感器、距离传感器和温度传感器。正是因为有了这些成熟的传感器技术的发展，我们可以在 Android 系统下设计开发出各种人性化，趣味性高的应用。

Android 平台包含各种用于监视环境的传感器选项。有了输入或模拟选项数组，以及高级计算和互联功能，Android 成为构建实际系统的最佳平台。

简单视图如图 6-6 所示。

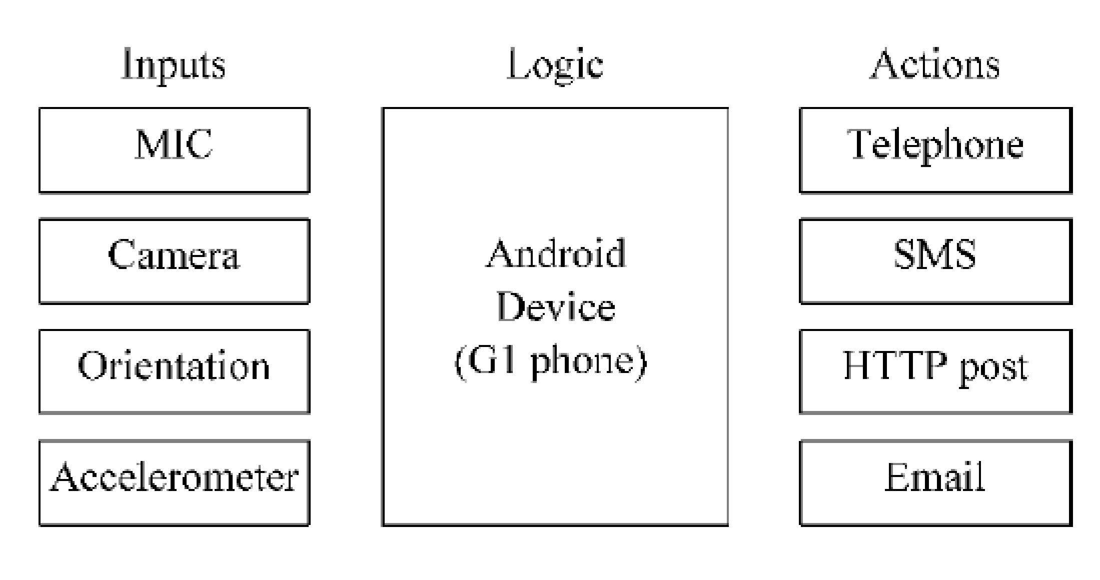


图 6-6　简单视图

该架构很灵活，应用程序逻辑可以划分为本地 Android 设备和服务器端资源(可以实现更大的数据库和计算功能)。例如，本地 Android 设备上录制的音轨可以 POST 到 Web

服务器，其中将根据音频模式数据库比较数据。很明显，这仅仅是冰山一角。

Android操作系统中内置了很多传感器，比如G1自带了一个非常实用的加速感应器(微型陀螺仪)，有了它，G1手机就支持重力感应、方向判断等功能，在部分游戏或软件中可以自动识别屏幕的横屏、竖屏方向来改变屏幕显示布局。下面是Android中支持的几种传感器：

- Sensor.TYPE_ACCELEROMETER：加速度传感器。
- Sensor.TYPE_GYROSCOPE：陀螺仪传感器。
- Sensor.TYPE_LIGHT：亮度传感器。
- Sensor.TYPE_MAGNETIC_FIELD：地磁传感器。
- Sensor.TYPE_ORIENTATION：方向传感器。
- Sensor.TYPE_PRESSURE：压力传感器。
- Sensor.TYPE_PROXIMITY：近程传感器。
- Sensor.TYPE_TEMPERATURE：温度传感器。

下面我们先来看一下无线传感器网络的特点：

(1) 节点更多，分布更密集，网络规模更大，为了在某个地理区域上进行监测，通常有成百上千甚至上万的节点被布置在该区域，如果单个节点或者局部几个节点出现故障是不会导致网络瘫痪的，所以利用节点之间的高度连接性可以保证系统的容错性和抗毁性。

(2) 资源更有限，由于受价格、体积和功耗的限制，传感器网中的传感器一般采用嵌入式处理器和存储器。这些传感器都具有计算能力，可以完成一些信息处理工作。但是，由于嵌入式处理器的能力和存储器的容量有限，因此传感器的处理能力也相当受限。

(3) 能量更有限，由于受到硬件条件的影响，无线传感器节点通常采用电池供电，电源能量更加受限。而多数传感器网又往往要求长时间工作，并且受到能量的影响，无线传感网络节点的通信距离更短，一般只有几十米，甚至更短。

(4) 由于传感器通常工作在环境恶劣的地区，更多地受到高大的障碍物如山脉、建筑物等以及自然环境如风雨雷电等的影响，一方面造成传感器之间的通信不能确保准确，另一方面可能使传感器出现长时间故障、甚至损毁。

(5) 多跳路由在无线传感器网络中，是基于数据为中心的路由，传感器网络节点没有一个全局性的标识，如IP地址。每个节点仅仅知道自己邻近节点的位置和标识，传感器网络通过相邻节点之间的相互协作来进行信号处理和通信，具有很强的协作性，而且数据传输具有很强的方向性。通常，查询信息是通过广播或多播的方式从观察者向网络内传感器传输，而探测结果信息则是由分布在各处的传感器节点向查询节点汇聚。

再来了解一下无线传感器网络中的关键技术。

(1) 能量挖掘。节点的能量成为无线传感器网络发挥效能的瓶颈。无线网络传感器节点的能量供应系统应根据自身特点进行设计，传感器节点功耗较低，但功耗变化范围比较大。如果利用能量挖掘技术从环境中挖掘能量，使节点具有能量补充的能力，这将从根本上解决节点的能量供给问题。典型方法是利用能量挖掘装置，可以挖掘各类能量如风能、太阳能、温差、振动等形式的能量。譬如，创业公司Perpetuum研制的PMG7微发电机，可以从一个100mg振动中产生5mW，3.3V的输出功率。因此巧妙地利用能量挖掘技术，从环境中挖掘能量，使节点具有能量补充的能力，并进一步结合能量管理及能量存储技术，才能在

根本上解决节点的能量供给问题。

(2) 能量管理。无线传感器网络节点多，覆盖范围大，工作环境复杂，能源无法替代，设计有效的策略延长网络的生命周期成为无线传感器网络的核心问题。休眠机制是节省能源的最有效方式之一，如何进行休眠调度而不影响传感器网络的正常运行十分重要。并且这个调度不能影响传感器网的功能，即网络的探测覆盖范围不能过分降低，否则这个网络可能无法达到探测目的。设计采用一种基于规则推理的大规模无线传感器网络智能能量管理算法。该算法的核心思想是根据被监测实体以往情况以及当前状态信息，通过基于规则的推理推测出下一个时间段内实体可能发生异常或者期待事件的区域，让监测该区域的传感器节点工作，监测其他区域的节点休眠，从而提高能量效率。

(3) 定位技术。定位技术主要指的是节点定位，即确定传感器的每个节点的相对位置或绝对位置。它是无线传感器网络研究领域中非常重要的一个研究方向，特别是军事应用的基础。WSNs 系统可以智能地选择一些特定的节点来完成任务，从而大大降低整个系统的能耗，提高系统的存活时间。根据节点位置估测机制，当前无线传感器网络的节点定位算法可分为基于距离的定位算法和距离无关的定位算法。由于要实际测量节点间的距离或角度，基于距离的定位机制通常定位精度相对较高，所以对节点的硬件也提出了很高的要求，典型的基于距离的定位算法分别利用 RSSI、TOA、TDOA、AOA 测距来定位节点。距离无关的定位机制无须测量节点间的绝对距离或方位，因而降低了对节点硬件的要求，使得节点成本更适合于大规模无线传感器网络，典型的距离无关的定位算法包含质心法、DV2Hop、Amorphous 和 APIT 算法。距离无关的定位机制的定位性能受环境因素的影响小，定位精度能够满足多数传感器网络应用的要求。

(4) 功率控制。所谓功率控制，就是要为每一个传感器节点选择合适的发射功率。有研究者将其简化为理想模型下发射范围分配问题进行分析。其结论是：在一维情况下，发射范围分配问题可以在多项式时间内解决；而在二维和三维情况下，发射范围分配问题是难的。这个结论从理论上告诉我们，试图寻找功率控制问题的最优解是不现实的，我们应寻找功率控制问题的实用解。目前功率控制研究大致形成了以下几个方向：

① 基于路由的功率控制。这个研究方向的基本观点是认为拓扑控制属于网络层，拓扑控制应该与路由协议相结合。其理由是：拓扑控制要保证网络的连通性，而网络的连通与否只有网络层才能检测。

② 基于节点度的功率控制。基于节点度的功率控制的核心思想是给定节点度的上限和下限，动态调整节点的发射功率，使得节点的度数落在上限和下限之间。

③ 基于方向的功率控制。基于方向的功率控制的基本思想是发射功率的选择要保证给定角度的、以节点为顶点的任意锥形区域内至少有一个邻居。基于方向的算法需要可靠的方向信息，因而需要很好地解决到达角度问题，节点需要配备多个有向天线。因而对传感器节点提出了较高的要求。

④ 基于邻近图的功率控制。基于邻近图的功率控制算法的基本思想是：设所有节点都使用最大发射功率发射时形成的拓扑图是 G，按照一定的邻居判别条件求出该图的邻近图，每个节点以自己所邻接的最远节点来确定发射功率。

(5) 睡眠调度。功率控制是通过降低节点的发射功率来延长网络的生存时间，并没有考虑空闲侦听时的能量消耗和覆盖冗余。无线通信模块在空闲侦听时的能量消耗与收发状

态时相当,覆盖冗余也造成了很大的能量消耗。所以只有使节点进入睡眠状态,才能大幅度地降低网络的能量消耗。对睡眠调度的研究工作大致可以分为平面型睡眠调度和层次型睡眠调度两大类。平面型网络中的所有节点都具有相同的功能、扮演相同的角色,否则就是层次性网络。

① 平面型睡眠调度。平面型睡眠调度的基本思想是让冗余节点进入睡眠状态。可大致分为基于覆盖冗余的睡眠调度和基于连通冗余的睡眠调度。基于覆盖冗余的睡眠调度的基本思想是:如果一个节点的感知区域已被其他节点覆盖,那么它就应该进入睡眠状态。这类算法一般需要精确的位置信息。

② 层次型睡眠调度。层次型网络睡眠调度的基本思想是:由簇头节点组成骨干网络,则其他节点就可以进入睡眠状态 层次型网络睡眠调度的关键技术是分簇。这些分簇算法可以大致分为随机分簇、基于地理位置分簇和基于最小支配集的分簇。

(6) 通信协议。以数据为中心,面向具体应用是无线传感网络与传统网络系统的主要区别之一。目前,对无线传感网络通信协议的研究主要集中在 MAC 层和网络层。其中,MAC 层主要集中在邻居节点间的单跳传输,而网络层更多是研究端到端的传输模式。无线传感网络的通信协议研究主要集中在以下三个方面:

① MAC 层协议。MAC 协议一般采用四种基本机制:竞争模式、时分复用、频分复用和码分复用。后两种的带宽利用率较低,消除干扰和碰撞的技术相对复杂,很少在无线传感网络中使用。

② 高可靠性的数据传输。无线传感网络的安全数据传输研究主要着眼于两个方面。一种是从维护路由安全的角度出发,利用经典的多径路由算法,将数据包分解成子数据包,通过多条路由传递,数据包在网关处重建,以增强系统的安全性和稳健性。另一种是将注意力集中在安全协议方面,提出了两种适用于无线传感网络的基本安全协议算法:SNEP 和 p~TESLA。SNEP 提供节点到网关之间的数据加密、认证与数据更新,p~TESLA 实现了网关广播数据的认证。

③ 网络层协议。无线传感网络的路由算法可以大致分为两类,基于指标的和不基于指标的。基于指标的路由算法根据指标的不同类型可以进一步分成基于坐标、基于相对位置和基于簇的。不基于指标的路由算法也可以进一步划分为按需路由方式和随机路由方式。

(7) 异构互连。为了更高效便捷地使用无线传感网络,需要将其接入互联网。网关是拓展互联网"最后一公里",将互联网延伸至无线传感网络,实现异构网络互联的关键部件。无线传感网络的网关具有多功能性、数据请求单向性和安全易损性等特点。目前普遍认同的接入方式是使用代理或 DTN 覆盖网。在代理方式中,代理以转发或者前端方式来工作,网关节点就是代理服务器。而 DTN 是采用覆盖网络进行网际互联的通用模型。DTN 中网关以转发器方式工作,使用代理转发的方式相当于是使用 DTN 的特例。使用多个网关可以避免单故障点问题。多网关接入技术和网状传感网络技术是相互渗透的技术。DTN 网络和网状传感网络都采用了覆盖网络的技术,DTN 中的覆盖网络相当于网状传感网络中的路由层和接入层,可以使用网状传感网络的技术。同样地,在网状传感网络中也可以使用多网关技术。

(8) 数据管理、查询与分析挖掘关键技术。无线传感网络以数据为中心的特点使其与数据库系统类似,即无线传感网络可视为一个支持感知数据查询的数据库。很多研究者采

用数据库研究方法来研究无线传感网络。无线传感网络数据管理技术的研究主要集中如下五个方面：

① 数据模型和查询语言。目前研究者对传统的关系模型和 SQL 语言进行了简单的修改和扩展。但还存在很多问题，如不能表达感知数据的不确定性、物理学和统计学特性、关联性、相关重要性等。

② 数据存储与索引技术。在无线传感网络内存储感知数据的方法和索引技术，对查询处理的性能具有重要影响。

③ 数据操作算法。无线传感网络数据操作算法的研究目前仅限于聚集操作算法。人们提出了两类聚集算法，即网络层聚集算法和应用层聚集算法。这些算法消耗大量存储资源并具有很大误差。

④ 查询处理技术。无线传感网络数据查询处理技术有基于感知数据流上处理连续查询的自适应技术、全网查询处理技术、模型驱动的查询处理方法等。

(9) 安全问题。安全问题是无线传感网络面临的一个严峻问题。无线传感网络大部分采用无线射频连接，无线电电磁的干扰使信噪比变差导致无法通信。而人为的干扰可以采用被动的方式监听网络中传送的数据包，还可以对监听到的数据包进行解析，主动发出入侵数据包以非法窃取或者修改某些重要信息，或者针对无线传感网络能量有限性的特点，发送大量无用的数据包导致能量耗尽而瘫痪。在无线传感网络的安全研究中“针对物理层主要采用的是高效加密算法”以及通过扩频通信减少电磁干扰，数据链路层的安全 MAC 协议，网络层安全路由协议以及应用层的密钥管理和安全组播。除了在路由协议上考虑安全机制，还可以在安全协议上加以研究。无线传感网络中的两种专用安全协议：SNEP 和 uTESLA。SNEP 的功能是提供节点到接收机之间数据的鉴权、加密、刷新，uTESLA 的功能是对广播数据的鉴权。当前对无线传感网络安全方面的研究集中在基于计算能力以及通信能力有限状态下的自适应安全机制。这类研究致力于加密与消息认证机制，及在密钥组管理。

下面再看一下 Android 特性，鉴于 Android 的开源以及制造商可对其自由定制的特点，其并没有固定的软硬件配置。然而，Android 本身支持如下特性：

存储——使用 SQLite，轻量级的关系数据库进行数据存储，第 6 章将对数据存储做详细讨论。

互联——支持 GSM/EDGE、IDEN、CDMA、EV-DO、UMTS、蓝牙(包括 A2DP 和 AVRCP)、WiFi、LTE，以及 WiMAX。

消息传递——支持 SMS 和 MMS。

Web 浏览器——基于开源的 WebKit，并集成 Chrome V8 的 JavaScript 引擎。

多媒体支持——支持以下媒体协议：H.263、H.264 (在 3GP 或 MP4 容器中)、MPEG-4 SP、AMR、AMR-WB (在 3GP 容器中)、AAC、HE-AAC (在 MP4 或 3GP 容器中)、MP3、MIDI、Ogg Vorbis、WAV、JPEG、PNG、GIF 以及 BMP。

硬件支持——加速度传感器、摄像头、数字式罗盘、接近传感器和全球定位系统。

多点触摸——支持多点触摸屏幕。

多任务——支持多任务应用。

Flash 支持——Android 2.3 支持 Flash 10.1。

数据链服务——支持作为有线/无线热点实现 Internet 连接共享。

Android 在管理 WSN 节点上将会发挥出很强大的功能。WSN 网络规模将会越来越大，自组网能力也将越来越强，WSN 节点计算存储能力差这一缺点也将不复存在。

一个系统的功能加的多了，运载这一系统所需要的体积就会大，能耗也必将多一些。而 WSN 中的节点一般要求体积要小一些，能耗的要求也是越小越好。

在目前对于节能问题的研究进展中，休眠机制是节省能源的最有效的方式之一。由于传感器节点监测事件的偶发性，没有必要让所有单元均工作在正常状态下，此时即可启用休眠模式，能自适应地休眠和唤醒，进行突发工作，节省能量。还可将所有功耗单元有机组合，形成不同状态，让传感器节点能根据需要在不同状态间切换。另外根据负载状态动态调节供电电压，形成一个闭环控制系统，也可达到节省能量的功效。总之，在满足系统要求的情况下，采用各种方法降低耗电量非常必要。

从 Android 2.1 系统开始就内置了一个性能非常强大的能源管理 widget。Android 作为一款嵌入式系统，本身就具有可裁剪性，随着 Android 在 WSN 中的应用和发展，将有一款新的越加适合 WSN 节点要求的系统诞生，一种新的能源管理措施也必将减少 WSN 节点的能耗。WSN 节点之间的通信也会更加安全。

节点定位是指确定传感器节点的相对位置或绝对位置，节点所采集到的数据必须结合其在测量坐标系内的位置信息才有意义。而 Android 系统就提供了非常强大的定位功能，只要使用三个接口和八大类就可以完成。GpsStatus. Listener：这是一个当 GPS 状态发生改变时，用来接收通知的接口。GpsStatus. NmeaListener：这是一个用来从 GPS 里接收 Nmea-0183(为海用电子设备制定的标准格式)信息的接口。LocationListener：位置监听器，用于接收当位置信息发生改变时从 LocationManager 接收通知的接口。Address：描述地址的类。Criteria：用于描述 Location Provider 标准的类，标准包括位置精度水平，电量消耗水平，是否获取海拔、方位信息，是否允许接收付费服务。GeoCoder：用于处理地理位置。GpsSatellite：和 GpsStatus 联合使用，用于描述当前 GPS 卫星的状态。GpsStatus：和 GpsStatus. Listener 联合使用，用于描述当前 GPS 卫星的状态。Location：用于描述位置信息。LocationManager：通过此类获取和调用系统位置服务。LocationProvider：用于描述 Location Provider 的抽象超类。

Android 系统内嵌了多种通信协议，WSN 网络节点通信会更加安全可靠，通信距离也会更加远。

Android 提供了 5 种方式存储数据：

(1) 使用 SharedPreferences 存储数据。

(2) 文件存储数据。

(3) SQLite 数据库存储数据。

(4) 使用 ContentProvider 存储数据。

(5) 网络存储数据。

应用 Android 系统的无线传感器节点将有能力存储和处理更加庞大的数据。

WSN 网络中会用到非常多的节点，这样成本就得控制得不能太高，嵌入 Android 系统以后，WSN 节点对硬件要求可能会高一些，成本也可能会高一些。

6.4.2 Android 在 WSN 中的应用前景

鉴于 Android 系统如此强大的传感器技术，Android 系统在 WSN 中应用无疑也必将是前途一片光明。随着 Android 系统在 WSN 中的应用与发展，WSN 节点也必将在原有优势下更加人性化，更加具有趣味性。无线传感器网络的许多基础性问题和关键技术，诸如能量有限性，计算存储能力，传输层和服务质量，覆盖率与部署，可靠数据传输，网络协议的统一标准等。尽管 Android 暂时还不能广泛应用于无线传感器网络中，但 Android 系统在 WSN 中的应用前景会十分广阔。

6.5 无线传感器网络的应用实例

无线传感器网络是大量的静止或移动的传感器以自组织和多跳的方式构成的无线网络，其目的是协作地感知、采集、处理和传输网络覆盖地理区域内感知对象的监测信息，并报告给用户。无线传感器网络体系结构如图 6-7 所示。

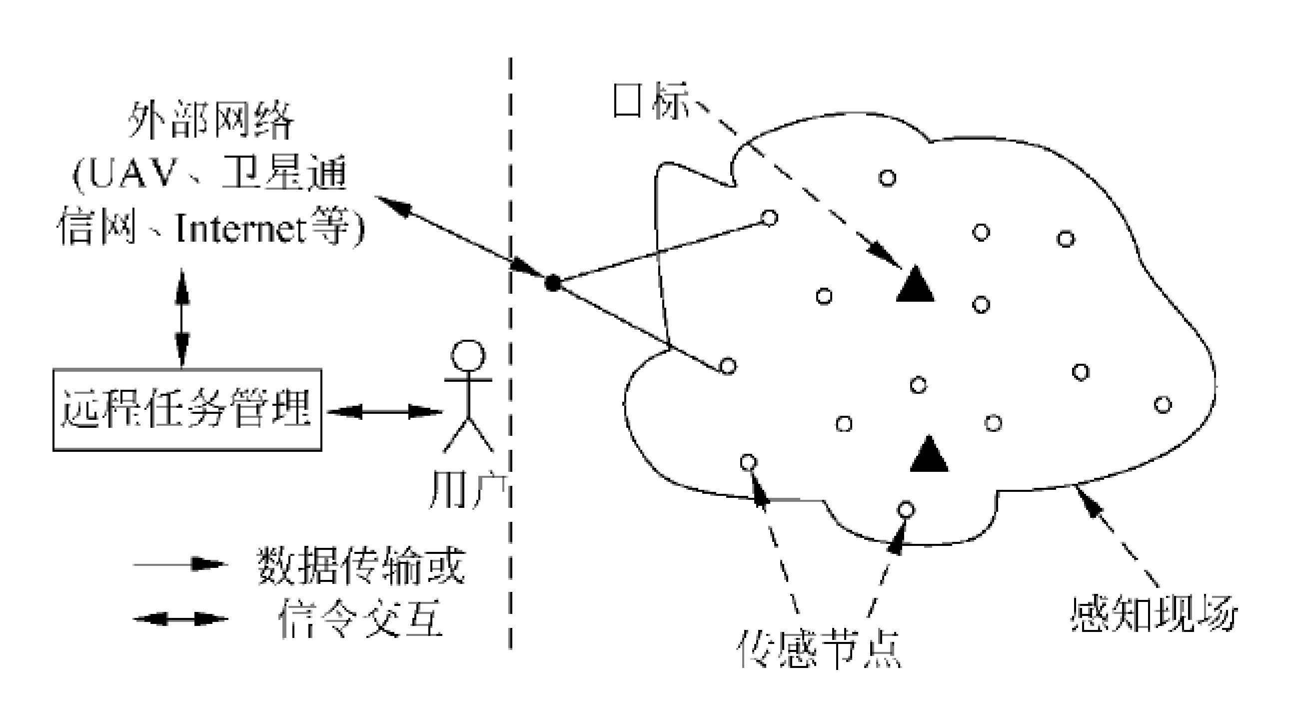


图 6-7 无线传感器网络的体系结构

它的英文是 Wireless Sensor Network，简称 WSN。大量的传感器节点将探测数据通过汇聚节点经其他网络发送给了用户。

6.5.1 传感器网络的特点

1. 大规模网络

为了获取精确信息，在监测区域通常部署大量传感器节点，传感器节点数量可能达到成千上万，甚至更多。传感器网络的大规模性包括两方面的含义：一方面是传感器节点分布在很大的地理区域内，如在原始大森林采用传感器网络进行森林防火和环境监测，需要部署大量的传感器节点；另一方面，传感器节点部署很密集，在一个面积不是很大的空间内，密集部署了大量的传感器节点。

传感器网络的大规模性具有如下优点：通过不同空间视角获得的信息具有更大的信噪比；通过分布式处理大量的采集信息能够提高监测的精确度，降低对单个节点传感器的精度要求；大量冗余节点的存在，使得系统具有很强的容错性能；大量节点能够增大覆盖的监测区域，减少洞穴或者盲区。

2. 自组织网络

在传感器网络应用中，通常情况下传感器节点被放置在没有基础结构的地方。传感器节点的位置不能预先精确设定，节点之间的相互邻居关系预先也不知道，如通过飞机播撒大量传感器节点到面积广阔的原始森林中，或随意放置到人不可到达或危险的区域。这样就要求传感器节点具有自组织的能力，能够自动进行配置和管理，通过拓扑控制机制和网络协议自动形成转发监测数据的多跳无线网络系统。

在传感器网络使用过程中，部分传感器节点由于能量耗尽或环境因素造成失效，也有一些节点为了弥补失效节点、增加监测精度而补充到网络中，这样在传感器网络中的节点个数就动态地增加或减少，从而使网络的拓扑结构随之动态地变化。传感器网络的自组织性要能够适应这种网络拓扑结构的动态变化。

3. 动态性网络

传感器网络的拓扑结构可能因为下列因素而改变：

(1) 环境因素或电能耗尽造成的传感器节点出现故障或失效；

(2) 环境条件变化可能造成无线通信链路带宽变化，甚至时断时通；

(3) 传感器网络的传感器、感知对象和观察者这三要素都可能具有移动性；

(4) 新节点的加入。

这就要求传感器网络系统要能够适应这种变化，具有动态的系统可重构性。

4. 可靠的网络

传感器网络特别适合部署在恶劣环境或人类不宜到达的区域，传感器节点可能工作在露天环境中，遭受太阳的暴晒或风吹雨淋，甚至遭到无关人员或动物的破坏。传感器节点往往采用随机部署，如通过飞机撒播或发射炮弹到指定区域进行部署。这些都要求传感器节点非常坚固，不易损坏，适应各种恶劣环境条件。

由于监测区域环境的限制以及传感器节点数目巨大，不可能人工“照顾”每个传感器节点，网络的维护十分困难甚至不可维护。传感器网络的通信保密性和安全性也十分重要，要防止监测数据被盗取和获取伪造的监测信息。因此，传感器网络的软硬件必须具有鲁棒性和容错性。

5. 应用相关的网络

传感器网络用来感知客观物理世界，获取物理世界的信息量。客观世界的物理量多种多样，不可穷尽。不同的传感器网络应用关心不同的物理量，因此对传感器的应用系统也有多种多样的要求。

不同的应用背景对传感器网络的要求不同，其硬件平台、软件系统和网络协议必然会有很大差别。所以传感器网络不能像 Internet 一样，有统一的通信协议平台。对于不同的传感器网络应用虽然存在一些共性问题，但在开发传感器网络应用中，更关心传感器网络的差异。只有让系统更贴近应用，才能做出最高效的目标系统。针对每一个具体应用来研究传感器网络技术，这是传感器网络设计不同于传统网络的显著特征。

6. 以数据为中心的网络

目前的 Internet 是先有计算机终端系统，然后再互联成为网络，终端系统可以脱离网络独立存在。在 Internet 中，网络设备用网络中唯一的 IP 地址标识，资源定位和信息传输依赖于终端、路由器、服务器等网络设备的 IP 地址。如果想访问 Internet 中的资源，首先要知

道存放资源的服务器 IP 地址。可以说目前的 Internet 是一个以地址为中心的网络。

传感器网络是任务型的网络，脱离传感器网络谈论传感器节点没有任何意义。传感器网络中的节点采用节点编号标识，节点编号是否需要全网唯一取决于网络通信协议的设计。由于传感器节点随机部署，构成的传感器网络与节点编号之间的关系是完全动态的，表现为节点编号与节点位置没有必然联系。用户使用传感器网络查询事件时，直接将所关心的事件通告给网络，而不是通告给某个确定编号的节点。网络在获得指定事件的信息后汇报给用户。这种以数据本身作为查询或传输线索的思想更接近于自然语言交流的习惯。所以通常说传感器网络是一个以数据为中心的网络。

例如，在应用于目标跟踪的传感器网络中，跟踪目标可能出现在任何地方，对目标感兴趣的用户只关心目标出现的位置和时间，并不关心哪个节点监测到目标。事实上，在目标移动的过程中，必然是由不同的节点提供目标的位置消息。

7. 网络协议栈

无线传感器网络多采用五层协议标准：应用层、传输层、网络层、数据链路层、物理层，与 Internet 协议栈的五层协议相对应。另外，协议栈还包括能量管理平台、移动管理平台和任务管理平台。这些管理平台使得传感器节点能够按照能源高效的方式协同工作，在节点移动的传感器网络中转发数据，并支持多任务和资源共享。各层协议和平台的功能如下：

- 物理层提供简单但健壮的信号调制和无线收发技术；
- 数据链路层负责数据成帧、帧检测、媒体访问和差错控制；
- 网络层主要负责路由生成与路由选择；
- 传输层负责数据流的传输控制，是保证通信服务质量的重要部分；
- 应用层包括一系列基于监测任务的应用层软件；
- 能量管理平台管理传感器节点如何使用能源，在各个协议层都需要考虑节省能量；
- 移动管理平台检测并注册传感器节点的移动，维护到汇聚节点的路由，使得传感器节点能够动态跟踪其邻居的位置；
- 任务管理平台在一个给定的区域内平衡和调度监测任务。

6.5.2 无线传感器网络技术发展现状

无线传感器网络（WSN）是信息科学领域中一个全新的发展方向，同时也是新兴学科与传统学科进行领域间交叉的结果。无线传感器网络经历了智能传感器、无线智能传感器、无线传感器网络 3 个阶段。智能传感器将计算能力嵌入到传感器中，使得传感器节点不仅具有数据采集能力，而且具有滤波和信息处理能力；无线智能传感器在智能传感器的基础上增加了无线通信能力，大大延长了传感器的感知触角，降低了传感器的工程实施成本；无线传感器网络则将网络技术引入到无线智能传感器中，使得传感器不再是单个的感知单元，而是能够交换信息、协调控制的有机结合体，实现物与物的互联，把感知触角深入世界各个角落，必将成为下一代 Internet 的重要组成部分。

6.5.3 基于 WSN 网络的应用系统发展现状

WSN 网络是面向应用的，贴近客观物理世界的网络系统，其产生和发展一直都与应用相联系。多年来经过不同领域研究人员的演绎，WSN 技术在军事领域、精细农业、安全监

控、环保监测、建筑领域、医疗监护、工业监控、智能交通、物流管理、自由空间探索、智能家居等领域的应用得到了充分的肯定和展示。2005 年,美国军方成功测试了由美国 Crossbow 产品组建的枪声定位系统,为救护、反恐提供有力手段。美国科学应用国际公司采用无线传感器网络,构筑了一个电子周边防御系统,为美国军方提供军事防御和情报信息。

中国,中科院微系统所主导的团队积极开展基于 WSN 的电子围栏技术的边境防御系统的研发和试点,已取得了阶段性的成果。

在环境监控和精细农业方面,WSN 系统最为广泛。2002 年,英特尔公司率先在俄勒冈建立了世界上第一个无线葡萄园,这是一个典型的精准农业、智能耕种的实例。杭州齐格科技有限公司与浙江农科院合作研发了远程农作管理决策服务平台,该平台利用了无线传感器技术实现对农田温室大棚温度、湿度、露点、光照等环境信息的监测。

在民用安全监控方面,英国的一家博物馆利用无线传感器网络设计了一个报警系统,他们将节点放在珍贵文物或艺术品的底部或背面,通过侦测灯光的亮度改变和振动情况,来判断展览品的安全状态。中科院计算所在故宫博物院实施的文物安全监控系统也是 WSN 技术在民用安防领域中的典型应用。

现代建筑的发展不仅要求为人们提供更加舒适、安全的房屋和桥梁,而且希望建筑本身能够对自身的健康状况进行评估。WSN 技术在建筑结构健康监控方面将发挥重要作用。2004 年,哈工大在深圳地王大厦实施部署了监测环境噪声和震动加速度响应测试的 WSN 网络系统。

在医疗监控方面,美国英特尔公司目前正在研制家庭护理的无线传感器网络系统,作为美国“应对老龄化社会技术项目”的一项重要内容。另外,在对特殊医院(精神类或残障类)中病人的位置监控方面,WSN 也有巨大应用潜力。

在工业监控方面,美国英特尔公司为俄勒冈的一家芯片制造厂安装了 200 台无线传感器,用来监控部分工厂设备的振动情况,并在测量结果超出规定时提供监测报告。西安成峰公司与陕西天和集团合作开发了矿井环境监测系统和矿工井下区段定位系统。

在智能交通方面,美国交通部提出了“国家智能交通系统项目规划”,预计到 2025 年全面投入使用。该系统综合运用大量传感器网络,配合 GPS 系统、区域网络系统等资源,实现对交通车辆的优化调度,并为个体交通推荐实时的、最佳的行车路线服务。目前在美国的宾夕法尼亚州的匹兹堡市已经建有这样的智能交通信息系统。

中科院上海微系统所为首的研究团队正在积极开展 WSN 在城市交通的应用。中科院软件所在地下停车场基于 WSN 网络技术实现了细粒度的智能车位管理系统,使得停车信息能够迅速通过发布系统推送给附近的车辆,大大提高了停车效率。

物流领域是 WSN 网络技术是发展最快最成熟的应用领域。尽管在仓储物流领域,RFID 技术还没有被普遍采纳,但基于 RFID 和传感器节点在大粒度商品物流管理中已经得到了广泛的应用。宁波中科万通公司与宁波港合作,实现基于 RFID 网络的集装箱和集卡车的智能化管理。另外,还使用 WSN 技术实现了封闭仓库中托盘粒度的货物定位。

WSN 网络自由部署、自组织工作模式使其在自然科学探索方面有巨大的应用潜力。2002 年,由英特尔的研究小组和加州大学伯克利分校以及巴港大西洋大学的科学家把 WSN 技术应用于监视大鸭岛海鸟的栖息情况。2005 年,澳洲的科学家利用 WSN 技术来探测北澳大利亚蟾蜍的分布情况。佛罗里达宇航中心计划借助于航天器布撒的传感器节点

实现对星球表面大范围、长时期、近距离的监测和探索。

智能家居领域是 WSN 技术能够大展拳脚的地方。浙江大学计算机系的研究人员开发了一种基于 WSN 网络的无线水表系统，能够实现水表的自动抄录。复旦大学、电子科技大学等单位研制了基于 WSN 网络的智能楼宇系统，其典型结构包括了照明控制、警报门禁，以及家电控制的 PC 系统。各部件自治组网，最终由 PC 将信息发布在 Internet 上。人们可以通过 Internet 终端对家庭状况实施监测。

WSN 在应用领域的发展可谓方兴未艾，要想进一步推进该技术的发展，让其更好地为社会和人们的生活服务，不仅需要研究人员开展广泛的应用系统研究，更需要国家、地区，以及优质企业在各个层面上的大力推动和支持。

6.5.4 无线传感器网络的应用

无线传感器网络可以用在生活中的各个方面，下面举几个例子，如图 6-8 所示。

1. 无线传感器网络在农业中的应用

近十年来，随着智能农业、精准农业的发展，泛在通信网络、智能感知芯片、移动嵌入式系统等技术在农业中的应用逐步成为研究的热点。

无线传感器网络在农业中的应用如图 6-9 和图 6-10 所示。



图 6-8 应用领域

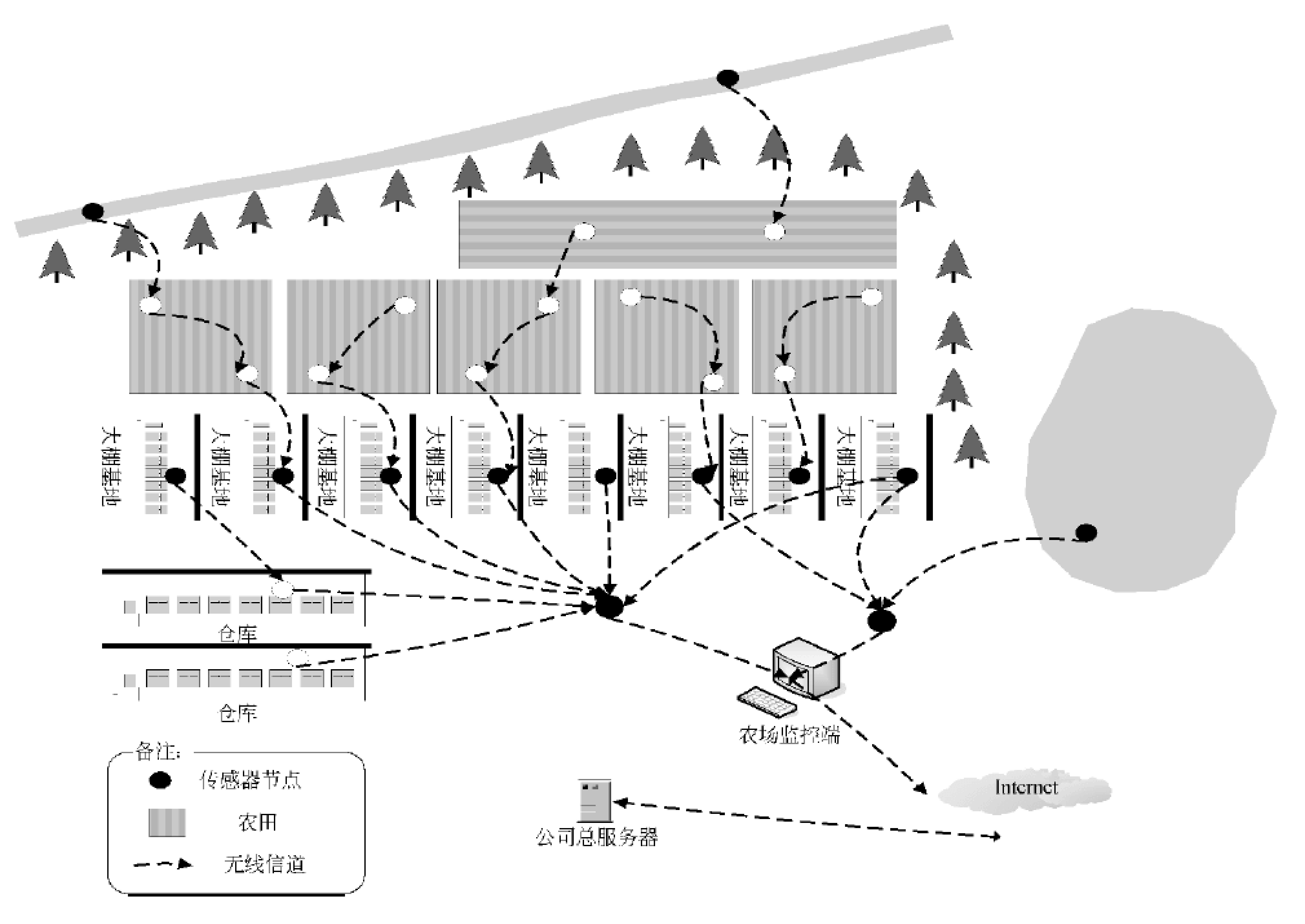


图 6-9 农田中的示意图

(1) 密集的无线传感器网络。

无线传感器网络是一种无中心节点的全分布系统。通过随机投放的方式，众多传感器

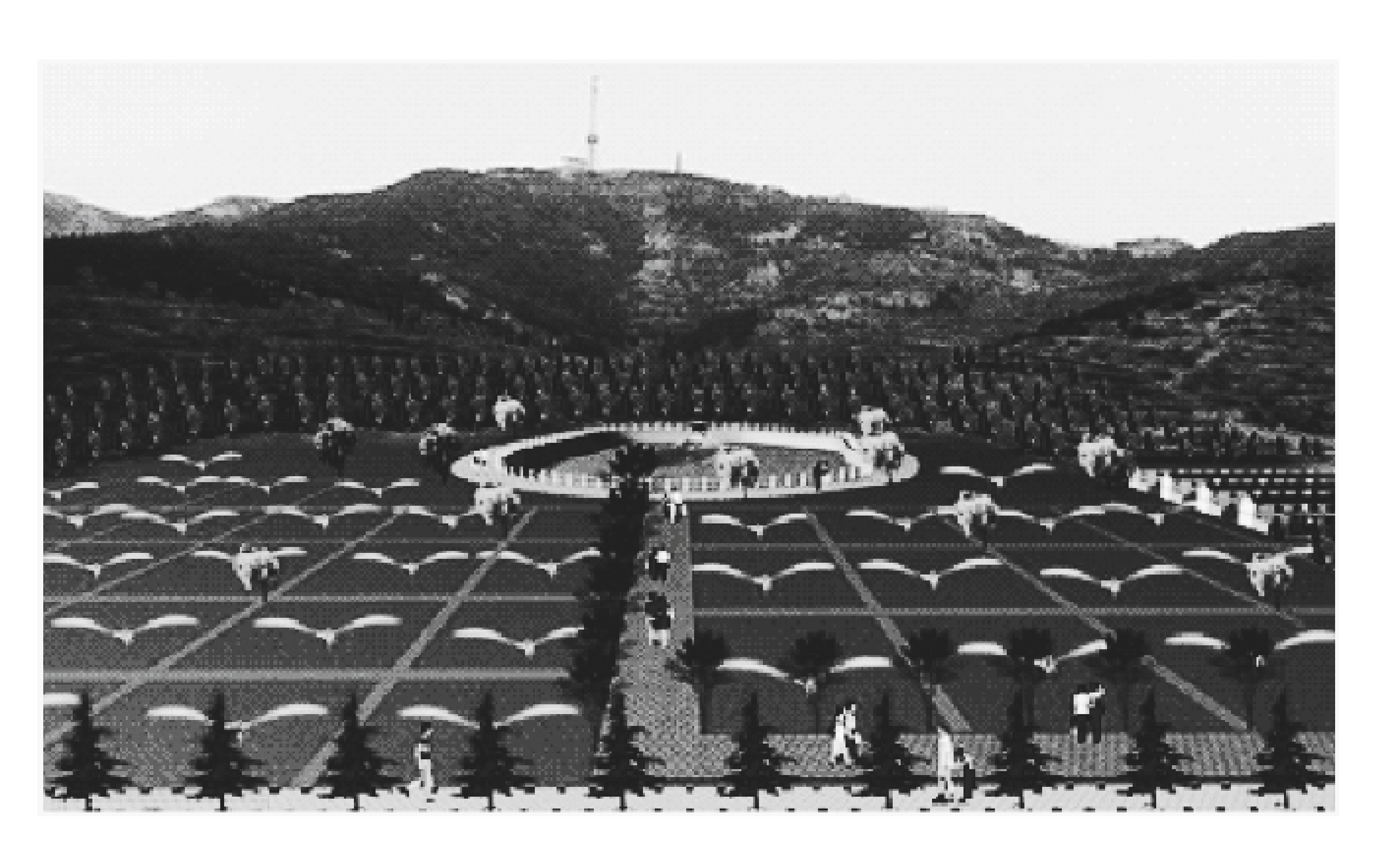


图 6-10　实际示意图

节点被密集部署于监控区域。这些传感器节点集成有传感器、数据处理单元、通信模块和能源单元，它们通过无线信道相连，自组织地构成网络系统。其目的是协作地感知、采集和处理网络覆盖区域中被监测对象的信息并发送给观察者。无线传感器网络集传感器技术、微机电系统（MEMS）技术、无线通信技术、嵌入式计算技术和分布式信息处理技术于一体，因其广阔的应用前景而成为当今世界上备受关注的、多学科高度交叉的热点研究领域。

在传统农业中，人们获取农田信息的方式都很有限，主要是通过人工测量，获取过程需要消耗大量的人力，而通过使用无线传感器网络可以有效降低人力消耗和对农田环境的影响，获取精确的作物环境和作物信息。

目前无线技术在农业中的应用比较广泛，但大都是具有基站星型拓扑结构的应用，并不是真正意义上的无线传感器网络。农业一般应用是将大量的传感器节点构成监控网络，通过各种传感器采集信息，以帮助农民及时发现问题，并且准确地确定发生问题的位置，这样农业将有可能逐渐地从以人力为中心、依赖于孤立机械的生产模式转向以信息和软件为中心的生产模式，从而大量使用各种自动化、智能化、远程控制的生产设备。

（2）无线传感器网络应用于温室环境信息采集和控制。

在温室环境里单个温室即可成为无线传感器网络一个测量控制区，采用不同的传感器节点和具有简单执行机构的节点（风机、低压电机、阀门等工作电流偏低的执行机构）构成无线网络来测量土壤湿度、土壤成分、pH 值、降水量、温度、空气湿度和气压、光照强度、CO_2浓度等来获得作物生长的最佳条件，同时将生物信息获取方法应用于无线传感器节点，为温室精准调控提供科学依据。最终使温室中传感器、执行机构标准化、数据化，利用网关实现控制装置的网络化，从而达到现场组网方便、增加作物产量、改善品质、调节生长周期、提高经济效益的目的。

（3）无线传感器网络应用于节水灌溉。

无线传感器网络自动灌溉系统利用传感器感应土壤的水分，并在设定条件下与接收器通信，控制灌溉系统的阀门打开、关闭，从而达到自动节水灌溉的目的。由于传感器网络多跳路由、信息互递、自组网络及网络通信时间同步等特点，使灌区面积、节点数量不受到限制，可以灵活增减轮灌组，加上节点具有的土壤、植物、气象等测量采集装置、通信网关的 Internet 功能与 RS 和 GPS 技术结合的灌区动态管理信息采集分析技术、作物需水信息采

集与精量控制灌溉技术、专家系统技术等构建高效、低能耗、低投入、多功能的农业节水灌溉平台。可在温室、庭院花园绿地、高速公路中央隔离带、农田井用灌溉区等区域，实现农业与生态节水技术的定量化、规范化、模式化、集成化，促进节水农业的快速和健康发展。

2008 年，湖南农业大学提出了一种基于无线传感器网络的农田自动节水灌溉构建方案，设计了一种无线传感器网络实现农田土壤湿度信息的实时采集和传输，通过灌溉控制器控制灌溉管网，分区域实时灌溉并调节土壤湿度，实现了精细农业所要求的时空差异性和水资源高效利用。

(4) 无线传感器网络应用于环境信息和动植物信息监测。

通过布置多层次的无线传感器网络检测系统，对牲畜家禽、水产养殖、稀有动物的生活习性、环境、生理状况及种群复杂度进行观测研究，也可用于对森林环境监测和火灾报警(平时节点被随机密布在森林之中，平常状态下定期报告环境数据，当发生火灾时，节点通过协同合作会在很短的时间内将火源的具体地址、火势大小等信息传送给相关部门)。同时也可以应用在精准农业中，来监测农作物中的害虫、土壤的酸碱度和施肥状况等。

在“十五”期间，国家 863 计划数字农业重大专项实现了农田信息采集技术的突破，推出了一批成本低、高性能的土壤水分和作物营养信息采集技术产品，基本解决了数字农业信息快速获取技术瓶颈问题。开展了农田水分、养分、作物长势、冠层生理与生态因子、品质、产量和虫害草害等信息采集关键技术研究，开发了具有自主知识产权的新型土壤水分传感器，研制了土壤和作物养分信息快速采集方法与新型配套仪器设备；在虫害与杂草动态监测系统的研究方面取得了重大进展，开发了基于称重传感器的高精度智能测产系统，解决了智能测产与谷物品质监测系统的精度难题；使我国农业信息快速获取迈出了新的步伐。

现代化温室和工厂化栽培调节和控制环境(控制温度、湿度、光照、喷灌量、通风等)培育各种秧苗，栽培各种果蔬和作物。在这个过程中，需要温度传感器、湿度传感器、pH 值传感器、光传感器、离子传感器、生物传感器、CO_2 传感器等检测环境中的温度、相对湿度、pH 值、光照强度、土壤养分、CO_2 浓度等物理量参数，通过各种仪器仪表实时显示或作为自动控制的参变量参与到自动控制中，保证农作物有一个良好的、适宜的生长环境。

在果蔬和粮食的储藏过程中，温度传感器发挥着巨大的作用，制冷机根据冷库内温度传感器的实时参数值实施自动控制并且保持该温度的相对稳定。气调库相比冷藏库是更为先进的贮藏保鲜方法，除了温度之外，气调库内的相对湿度(RH)、O_2 浓度、CO_2 浓度、乙烯(C_2H_4)浓度等均有相应的控制指标。控制系统采集气调库内的温度传感器、湿度传感器、O_2 浓度传感器、CO_2 浓度传感器等物理量参数，通过各种仪器仪表适时显示或作为自动控制的参变量参与到自动控制中，保证有一个适宜的贮藏保鲜环境，达到最佳的保鲜效果。

在作物的生长过程中还可以利用形状传感器、颜色传感器、重量传感器等来监测物的外形、颜色、大小等，用来确定物的成熟程度，以便适时采摘和收获；可以利用二氧化碳传感器进行植物生长的人工环境的监控，以促进光合作用的进行。例如，塑料大棚蔬菜种植环境的监测等；可以利用超声波传感器、音量和音频传感器等进行灭鼠、灭虫等；可以利用流量传感器及计算机系统自动控制农田水利灌溉等。

生物技术、遗传工程等都成为良种培育的重要技术，在这其中生物传感器发挥了重要的作用。农业科学家通过生物传感器操纵种子的遗传基因，在玉米种子里找到了防止脱水的基因，培育出了优良的玉米种子。此外，监测育种环境还需要温度传感器、湿度传感器、光传

感器等；测量土壤状况需用水分传感器，吸力传感器、氢离子传感器、温度传感器等；测量氮磷、钾各种养分需要用各种离子敏传感器。

在动物饲养中也有传感器应用，如有可用来测定畜、禽肉鲜度的传感器。它可以高精度地测定出鸡、鱼、肉等食品变质时发出的臭味成分二甲基胺(DMA)的浓度，其测量的最小浓度可以达到 1ppm，利用这种传感器可以准确地掌握肉类的鲜度，防止腐败变质。也有用来检测鸡蛋质量的传感器。

(5) 挖掘潜在应用。

物联网在农业领域具有远大的应用前景。

在农田、果园等大规模生产方面，如何把农业小环境的温度、湿度、光照、降雨量等，土壤的有机质含量、温湿度、重金属含量、pH 值等，以及植物生长特征等信息进行实时获取传输并利用，对于科学施肥、灌溉作业来说具有非常重要的意义。

在生鲜农产品流通方面，需要对储运环境的温度和农产品的水分进行控制，环境温度过高可能会发生大批农产品的腐烂，水分不足品质会受到影响，在这个环节要借助物联网的帮助。

还有一类具有典型意义的应用是工厂化健康养殖作业，需要通过物联网技术实现畜禽、水产养殖环境的动态监测与控制。

2. 无线传感器网络在工业中的应用

无线传感器网络在工业中的应用如图 6-11 和图 6-12 所示。

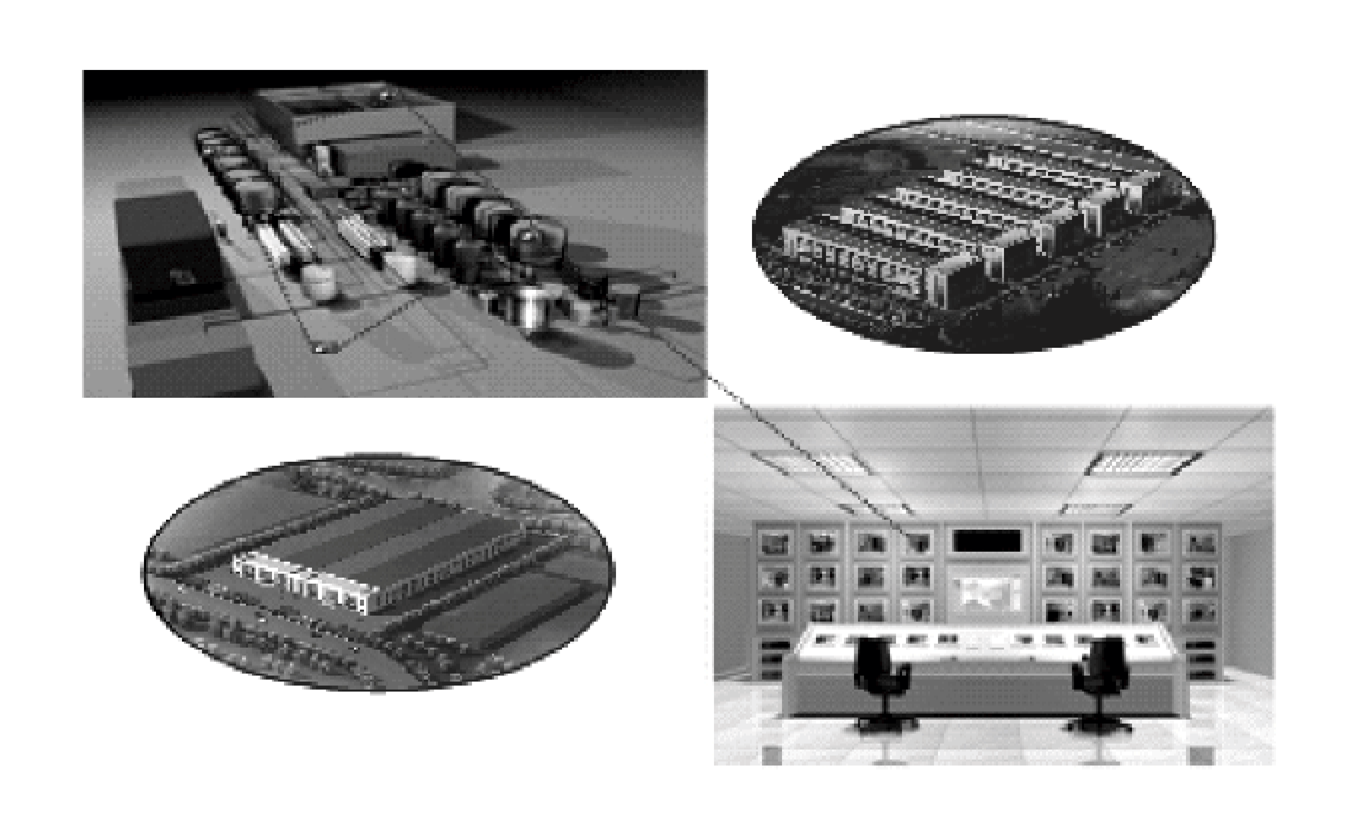


图 6-11　无线传感器网络在工业中的应用示意图 1

无线传感器网络是由在空间上相互离散的众多传感器相互协作组成的传感器网络系统。通常被用来监测在不同地点的物理或者环境参量，例如光、温度、湿度、声音、振动、压力、运动或者污染等。无线传感器网络的发展起初是源于军队应用的需要，例如战区战场监控。然而，无线传感器网络现在被更广泛地用于民用以及工业领域，包括自然和人居环境监控、医疗监护、家用电器自动化、交通控制、气象监测等领域。

在无线传感器网络中的每一个节点都配有无线电发射和接收装置，能够和网络中的其他节点进行通信。此外，每一个节点都有自己的微控制器，能量源(通常是电池)。单个的传

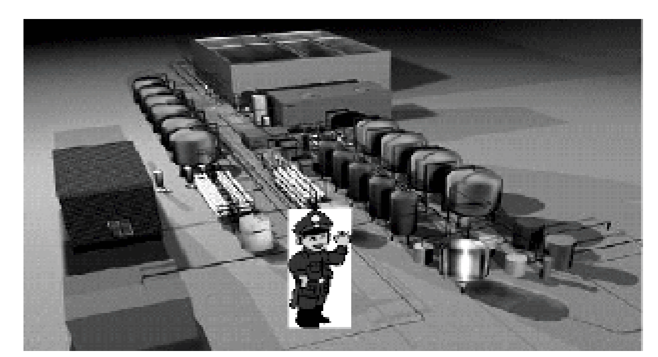




图 6-12　无线传感器网络在工业中的应用示意图 2

感器节点的形状有很多种，有可能像个鞋盒子，也有可能像一颗谷粒。无线传感器网络的成本也差别很大，从数百万美元到几分钱，都有可能，这取决于网络的规模和复杂程度。传感器的尺寸和费用预算的限制也会导致传感器网络性能的局限，例如存储空间、计算性能、网络速度以及带宽。

无线传感器网络通常会组成一个无线自组织网络，即每一个传感器节点都支持多跳路由算法(每一个节点都能作为网络内其他节点的数据中继站，将传感器数据传输到基站)。

1）污水处理厂的水质监控

一个无线传感器节点上连接一个液位传感器，一个 pH 值传感器，一个流量计和一个电导率传感器。许多个这样的节点被放置在污水处理池的各个位置，用来监测每个处理池中各处的污水处理状况，通过节点自动组成的智能网络，将各个节点的数据回传到远程控制中心的机房终端显示器上。由于无线传感器摆脱了信号电缆的束缚，因此可以任意安放在各个位置，并且可以随时对位置进行更换而不影响这个系统的监控，同时因为没有电缆，也不用担心发生漏电事故。

2）建筑物强度结构健康监测

在地震中，对人的生命财产安全造成最大伤害的就是建筑物的倒塌。而现今大都市中，摩天大楼林立，在汶川大地震中，北京地区也有震感，华贸、国贸等高层写字楼均有晃动，大量人员有不适感，但直至通过广播、网络确认地震发生后，写字楼人员方开始撤离。如果震中发生在北京附近，这几分钟的迟疑就会带来高层写字楼数千生命的消逝，而北京至少拥有数百栋高层写字楼。

加速度计依然是监测建筑物的最简单有效方式。美国加州大学伯克利分校对旧金山金门大桥部署过建筑健康监测系统。其本意是用来检测桥体在风力作用下的各个关键受力点的振动状况，整体数据建模后就可以分析出桥体受损老化严重的部分从而进行有针对性的修补。

桥体和高层建筑有一个共同的特点，就是建筑结构极其敏感，因此其前端的测量点部署很难采用有线方式，否则极易损害建筑结构受力。

而无线技术，特别是不需要供电的低功耗无线技术，在解决建筑物健康监测前端 100 米数据获取中具有极其重大的意义。节点具有无线能力，体积较为小巧，可以很容易地安装在建筑物的关键受力点上，而不影响建筑物外观。具有低功耗能力，节点一经部署不需要频繁

更换。省去了复杂耗时的布线操作，只要打开节点开关，位于建筑物监控中心的接收终端就可以实时获取数据，与建筑报警系统联动后，一旦探测到可能威胁到建筑物的震动信息，立即发出报警通知建筑物内人员立即撤离。平时该系统收集的数据还可以用来监测建筑物老化状况，为建筑物维护提供辅助决策信息。

3）桥梁强度结构健康监测

近年来中国连续发生的几起桥梁垮塌事故使人们加强了对桥梁健康状况的重视，相关部门也意识到桥梁并不是一个一次性投入后就可以放任不管的公共交通基础设施，它的健康状况同样关乎人民的生命和国家经济的安全。利用传统的传感器和测量监控设备对桥梁健康进行监测需要在整个桥面上铺设信号和供电电缆，不仅浪费资源而且铺设起来工程复杂庞大，特别是像杭州湾大桥和港珠澳大桥这种长距离、大跨度的巨型桥梁。而使用无线传感器网络来进行桥梁健康监测，就显得简单方便得多了，只需要将无线传感器节点固定在桥梁的关键受力点处，并沿桥布置，各个节点就会自动组织形成网络并且回传相关的测量数据。另外，我们还开发了专门的无线传感平台，可以连接多种传感器。这样，即使在建设桥梁时预埋进桥体的传感器也可以通过无线节点进行监测。不但充分利用了资源，还节省了技术升级所带来的成本投入。

4）原油输油管线温度压力监测

随着世界能源消耗量的增加，各个国家越来越依赖于从别国进口能源，特别是石油和天然气等基础能源原料，而石油输油管线又是其中最为重要的一种。但是世界原油储藏分布在一些高纬度国家，比如俄罗斯和加拿大。而原油的属性又决定它必须达到一定温度才能被稀释，具有良好的黏度特性和流动性。同时，又本着节约能源的目的，我们又要把加热的温度控制在一个合理的范围内，因此，就需要对石油输油管道的温度进行监测。在确保原油能够达到良好流动性的同时降低加热的能耗。在原有管道沿线布设无线传感器网络是最为经济合理的方案，这样可以大大节省铺设信号和供电电缆的成本。同时又能实现数据的网络智能传输。同时在节点上连接压力计又能够监测输油管线是否发生泄漏和偷盗事故，并及时报告事故位置。

5）通过无线网络监控室外储存罐内液位，并提供电源转换功能

与无线网络相连的传感器监控储存罐内的液位。因为只有 120V 交流可用，所以无线网络节点能够转换成可用的无线电源。无线网络节点是基于可抵御各种天气环境的设计，避免了耗时的维护工作。

6）地质灾害的预防监测-山体滑坡

香港由于存在大量山地地貌，城市居民人口众多，要求土地必须保持较高的利用率，因此大量建筑和道路都位于山区附近。由于地处中国南方，地理位置决定了该地区降雨量常年偏高，尤其在每年夏季的梅雨季节，会出现大量的降水。不稳定的山地地貌在受到雨水侵蚀后，容易产生山体滑坡现象，对居民生命财产安全造成巨大的威胁。

过去数十年内在某些极其危险地域发生了多次山地滑坡现象，因此政府部门试图部署一种灵活稳定的系统对山体滑坡进行监测和预警。该市政府部门尝试部署过多套有线方式的监测网络，但是由于监测区域往往为人迹罕至的山间，缺乏道路，野外布线，电源供给等都受到限制，使得有线系统部署起来非常困难。此外有线方式往往采用就近部署数据记录仪的方式记录采集数据，需要专人定时前往监测点下载数据，系统得不到实时数据，灵活性较

差。有关方面的专家为此专门制定了应用无线传感器网络的解决方案。

山体滑坡的监测主要依靠两种传感器：液位传感器以及倾角传感器。在山体容易发生危险的区域，将会沿着山势走向竖直设置多个孔洞。在每个孔洞的最下端部署一个液位传感器，在不同深度部署数个倾角传感器。由于该地区的山体滑坡现象主要是由雨水侵蚀产生的，因此地下水位深度是标识山体滑坡危险度的第一指标。该数据由部署在孔洞最下端的液位深度传感器采集并由无线网络发送。

通过倾角传感器我们可以监测山体的运动状况，山体往往由多层土壤或岩石组成，不同层次间由于物理构成和侵蚀程度不同，其运动速度不同。发生这种现象时我们部署在不同深度的倾角传感器将会返回不同的倾角数据。在无线网络获取到各个倾角传感器的数据后，通过数据融合处理，专业人员就可以据此判断出山体滑坡的趋势和强度，并判断其威胁性大小。山体滑坡在地震之后的灾区随处可见，尤其是交通要道两侧的山体滑坡对救援进度更是会造成巨大的威胁，相信无数人仍然记得在听到理县到汶川的生命线在打通后不到一天的时间就又因山体滑坡而中断时那种揪心的感觉。

6.6 WSN 的安全性问题

无线传感器网络是由部署在监测区域内大量具有特定功能的微型传感器节点通过无线通信方式形成的自组织网络，其目的是协同地感知、采集和处理网络覆盖区域内被监测对象的信息，并将该信息传送到观察者。作为一种新的计算模式，WSN 可以使观察者随时随地实时地监测网络覆盖区域内物理环境的变化，以获取大量翔实可靠的信息，从而真正实现“普适计算”的理论。这些特性使得 WSN 的应用范围非常广泛，涉及军事应用、工业监视与控制、环境监测、医疗监护、智能家居/建筑、仓储/物流管理、交通控制管理、精细农业、消费电子、太空探索、反恐、救灾等诸多领域。各种基于 WSN 的应用系统均对安全性提出了很高的要求。但是，WSN 节点具有计算能力、通信能力、电源能量和存储空间等资源严重受限的特点，使得设计并部署 WSN 安全机制，实现 WSN 的安全通信，面临着诸多困难和挑战。目前，安全性问题已经成为 WSN 获得实际应用的主要障碍之一。因此，开展 WSN 安全性问题研究，具有重大的理论意义和应用价值，国内外许多大学、研究机构和企业对此高度重视，纷纷加入到该研究行列。

6.6.1 无线传感器网络的安全维

WSN 具有通信能力受限、电源能量受限、计算速度受限、存储空间受限、传感器节点配置密集和网络拓扑结构灵活多变等特点，这些特点决定了 WSN 比传统网络更易遭受各种攻击。因此，一种比较完善的安全解决方案应该满足 WSN 的如下安全需求：

- 数据机密性
- 认证
- 数据完整性
- 有效性
- 访问控制
- 不可抵赖性

- 通信安全
- 数据保密性

1. 无线传感器网络的威胁模型

在 WSN 中,存在各种各样的安全威胁,这些安全威胁可以概括为 2 种类型,涉及节点的威胁和涉及路由选择的威胁。其中,涉及节点的威胁主要包括节点泄密、窃听、传感数据保密、DoS 攻击、恶意的网络等;涉及路由选择的威胁主要包括哄骗、变更和重放路由选择信息、选择性转发、Sinkhole 攻击、Sybil 攻击、Wormhole 攻击、HELLO 扩展攻击(flood attacks)、应答哄骗等。上述 2 类威胁为 WSN 所特有,统称为 WSN 的特殊威胁。

此外,[ITU-T X. 800]和[ITU-T X. 805]定义了传统网络所面临的各种威胁,其中,某些威胁适合于 WSN,称为 WSN 的一般威胁。

概括地说,WSN 威胁模型具有如表 6-4 所示的结构。

表 6-4 WSN 的威胁模型

一 般 威 胁	特 殊 威 胁	
	涉及节点的威胁	涉及路由选择的威胁
信息破坏	节点泄露	哄骗、变更和重放攻击
信息讹误或信息篡改	窃听	选择性转发
信息行窃或信息的损失	传感数据保密	Sinkhole 攻击
信息泄露	DoS 攻击	Sybil 攻击
服务中断	恶意网络	Wormhole 攻击 HELLO 扩散攻击 应答哄骗

2. 无线传感器网络安全维到威胁的映射

在讨论 WSN 的安全部署与安全机制之前,需要研究各类消息交换过程中安全维与安全威胁之间的映射关系。

在 WSN 中,一般威胁包括:信息破坏、信息讹误、信息行窃、信息泄露、服务中断等。安全维到一般威胁的映射如表 6-5 所示。在表 6-5 中,字母“Y”表示它所在列的安全性威胁能够被对应安全维所阻止。

表 6-5 安全维到一般威胁的映射

安全维	一 般 威 胁				
	信息破坏	信息讹误	信息行窃	信息泄露	服务中断
数据机密性			Y	Y	
认证	Y	Y	Y	Y	
数据完整性	Y	Y			
有效性	Y				Y
访问控制	Y	Y	Y	Y	
不可抵赖性	Y	Y	Y	Y	Y
通信安全			Y	Y	
数据保密性		Y	Y	Y	

3. WSN中安全维到特殊威胁的映射

(1)传感器节点之间消息交换过程中安全维到安全威胁的映射。在传感器节点之间消息交换过程中可能遭受如下威胁：节点泄密、窃听、传感数据保密、DoS攻击、恶意的网络、重放攻击等。安全维到这些安全威胁的映射如表6-6所示。在表6-6中，字母"Y"表示它所在列的安全性威胁能够被对应安全维所阻止。

表6-6 传感器节点之间消息交换过程中安全维到威胁的映射

安全维	特殊威胁					
	节点泄露	窃听	传感数据保密	DoS攻击	恶意网络	重放攻击
数据机密性	Y	Y	Y		Y	
认证				Y	Y	Y
数据完整性						
有效性				Y		
访问控制						
不可抵赖性				Y		
通信安全	Y	Y	Y		Y	
数据保密性			Y			

(2) 汇节点与节点之间消息交换过程中安全维到安全威胁的映射。在WSN系统中，传感数据通过汇节点转发到主干网络，再经过主干网络传送至用户网络。另一方面，汇节点也向传感器网络发布广播报文。在这个过程中，可能遭受如下威胁：信息破坏、信息讹误、信息行窃、信息泄露、服务中断、DoS攻击等。安全维到这些安全威胁的映射如表6-7所示。在表6-7中，字母"Y"表示它所在列的安全性威胁能够被对应安全维所阻止。

(3) 路由消息交换过程中的安全维到安全威胁的映射。节点之间路由消息交换的主要目标是建立能源有效性路径，形成可靠数据转发机制，延长WSN系统生命周期。在路由消息交换过程中，可能受到如下安全威胁：重放攻击、选择性转发、Sinkhole攻击、Sybil攻击、Wormhole攻击、HELLO扩散攻击、应答哄骗、DoS攻击等。表6-8列出了节点之间路由消息交换过程中安全维到安全威胁的映射。在表6-8中，字母"Y"表示它所在列的安全性威胁能够被对应安全维所阻止。

表6-7 汇节点与节点之间消息交换过程中安全维到威胁的映射

安全维	一般威胁			特殊威胁		
	信息破坏	信息讹误	信息行窃	信息泄露	服务中断	DoS攻击
数据机密性			Y	Y		
认证		Y	Y	Y		
数据完整性	Y	Y				
有效性	Y				Y	Y
访问控制	Y	Y	Y	Y		
不可抵赖性	Y	Y	Y		Y	Y
通信安全	Y	Y	Y	Y		
数据保密性			Y	Y		

表 6-8　路由消息交换过程中安全维到威胁的映射

安全维	特殊威胁							
	重放攻击	选择性转发	Sinkhole 攻击	Sybil 攻击	Wormhole 攻击	HELLO 泛洪攻击	应答哄骗	DoS 攻击
数据机密性		Y	Y		Y			
信息认证	Y					Y	Y	Y
实体认证	Y	Y	Y	Y	Y	Y	Y	Y
数据完整性		Y		Y		Y		
有效性								Y
访问控制	Y	Y	Y	Y	Y	Y		
不可抵赖性					Y	Y		
通信安全		Y	Y		Y			
数据保密性		Y	Y		Y	Y		

6.6.2　无线传感器网络安全性框架

1. 无线传感器网络安全维与安全功能之间的关系

安全功能用于满足安全需求，分析了 WSN 网络安全维与安全性功能集之间的关系，如表 6-9 所示。在表 6-9 中，字母“Y”表示其所在列的安全功能无法保障所在行的安全业务；字母“K”表示其所在列的安全功能可以增强所在列的安全业务；字母“X”表示其所在行的安全功能可以提供所在行的安全业务。

2. 无线传感器网络安全性框架

6 种安全机制的基本含义如下。

1）加密或解密

攻击者使用大功率接收机和精心设计的天线，对 WSN 系统进行窃听，能够轻易地窃取各种感兴趣的信息，包括节点的物理位置信息、消息 ID、时间戳以及数据包中其他字段的信息。在 WSN 中，使用强有力的加密或解密技术，可以实现各种通信数据或存储数据的机密性，从而把因窃听造成的损失降低到最低。具体地说，在数据链路层，加密数据信息可以有效防止攻击者对无线链路的窃听、篡改以及重放攻击。在网络层，加密路由信息可以有效防止攻击者实施篡改路由以加入网络、Sybil 攻击、重放攻击以及应答哄骗等。此外，WSN 系统还需要实施各种认证（例如信息认证、实体认证）和密钥管理来保证数据的完整性、有效性和机密性等，各种认证机制和密钥管理都依赖于加密或解密技术。可见，加密或解密是保证 WSN 系统进行安全通信的基础。但是，考虑到 WSN 中，节点的各种资源受限的特点，应该采用轻量化加密或解密算法。

2）认证

认证分为信息认证和实体认证。信息认证主要是确认信息源的合法身份以及保证信息的完整性，防止非法节点发送、伪造和篡改信息。实体认证是防止非法节点随意加入路由的重要措施。通常，实体认证可以采用 2 种方法：直接确认和间接确认。在直接确认中，一个可信节点直接检验新加入节点对新加入节点的合法性。在间接确认中，则需要另外一个可信的节点对新加入节点的合法性进行担保。考虑到 WSN 中，节点的各种资源受限的特点，应该采用轻量化认证算法。

表 6-9　安全维与安全功能之间的关系图

安全需求		安全功能									
		加密	信息认证	实体认证	数学签名	密钥管理	完整性	访问控制		有效性	
								物理	技术	物理	技术
数据机密性	通信数据	Y				Y		K			
	存储数据	Y				Y		K			
认证	实体认证			Y		Y					
	信息认证		X	X	X	Y					
数据完整性	通信数据		X	X	X	Y	X				
	存储数据		X		X	Y	X				
有效性	通信数据							X		X	Y
	存储数据		X	X		Y			K		Y
访问控制	通信数据	K				K		K			
	存储数据	K	Y	Y	X	Y		K	Y		
不可抵赖性					Y	Y					
通信安全		K	X	X		Y	X	K	Y		K
数据保密性	通信数据	Y									
	存储数据	Y	X	X	X	Y		K	Y		

3）安全广播与组播

在安全广播与组播中，使用逻辑密钥分级结构或逻辑密钥树来有效分配密钥。标准的逻辑密钥分级结构包括根和叶子。中央的密钥分配中心称为密钥分级结构的根，它负责分配整个网络的密钥；每个节点则称为叶子。在密钥分级结构中，内部节点包含密钥，这些密钥用于密钥的更新过程。在逻辑密钥树中，节点加入时需要进行验证，节点离开时则需要删除相应的枝。逻辑密钥树提供节点验证机制和枝删除策略。

4）安全路由

在 WSN 中，路由和数据转发是基本服务。路由安全是网络安全的一个重要手段和措施。网络研究人员必须设计安全路由协议来抵御各种威胁，保障传感器网络的安全。设计安全路由时，可以考虑融合下面几个安全机制：

(1) 在路由设计中加入容侵策略。具体实现方法可以参照INSENS。

(2) 在路由设计中加入广播半径限制。基本思路是:对每个节点的数据发送半径进行限制,使该节点智能对这个半径区域内的节点发送数据,而无法对整个网络广播。在路由设计中加入广播半径限制后,可以有效避免具有高功率,高能量的攻击者在整个网络区域不断发送数据包,从而一定程度上可以抵御DoS攻击和扩散攻击,特别是HELLO扩散攻击。

(3) 在路由设计中加入信任模型。在路由设计中加入信任模型可以有效对抗Wormhole攻击和Sinkhole攻击,具体实现方法可以参照DSR-mod。

5) 分级体系结构

分级体系结构的基本含义表现在:构造树状路由来实现安全广播和安全组播。通过建立密钥分发树把节点标志出根、叶子的等级,对节点实施分级管理。这样做的好处是责任区域的界限清晰,在一定程度上可以有效抵御扩散法攻击,而且在网络遭受其他攻击时,能够加速发现问题节点。

6) 基于策略的方法

基于策略的方法主要用于解决保密性问题。访问控制和身份验证都是建立在保密策略的基础之上。基于策略的方法在实现形式上比较灵活,常见的实现形式包括在路由设计中加入容侵策略,冗余路由策略,消息优先级策略,数据时新性鉴别策略等。

3. 结论

在WSN真正投入实际应用之前,必须为其设计并实现轻量化的安全机制。虽然,国内外已有众多研究人员开展相关研究,但是进展甚微。主要的困难来自于WSN的特征及其应用背景,尤其是传感器节点的计算能力、通信能力、电源能量和存储空间等资源严重受限的实现,使得设计并部署WSN安全机制,面临着许多严峻的挑战。论文在充分吸收国内外已有先进理论成果的基础上,针对WSN的特点和应用背景,提出了一个适合于WSN的安全性框架,同时对实现该安全性框架的相关安全机制和安全技术进行了深入的分析。该安全性框架集成在WSN的轻量化协议,能够有效抵御各种恶意攻击,保障WSN的通信安全。

第 7 章　基于 Android 的技术开发实例

7.1　实例 1：打电话

1. 创建一个 Android 项目 Phone

创建方法与过程这里不再演示。

2. PhoneActivity.java 文件

```
package jiao.jiao;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class PhoneActivity extends Activity {
    /** Called when the activity is first created. */
  private EditText editText;
  private Button button;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    editText = (EditText)this.findViewById(R.id.editText);
    button = (Button)this.findViewById(R.id.call);
    button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            String phoneNum = editText.getText().toString();
        Intent intent = new Intent(Intent.ACTION_CALL,Uri.parse("tel:" + phoneNum));
            PhoneActivity.this.startActivity(intent);
        }
    });
     }
}
```

3. main.xml 文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
      >
    <TextView
              android:layout_width = "wrap_content"
          android:layout_height = "wrap_content"
          android:text = "@string/hello"
           />
  <EditText
              android:layout_width = "fill_parent"
          android:layout_height = "wrap_content"
          android:id = "@ + id/editText"
              />
          <Button
              android:layout_width = "wrap_content"
          android:layout_height = "wrap_content"
          android:id = "@ + id/call"
          android:text = "@string/call_button"
              />
   </LinearLayout>
```

4. strings. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
<resources>
    <string name = "hello">打电话</string>
    <string name = "app_name">Phone</string>
    <string name = "call_button">打电话</string>
</resources>
```

5. 加一个打电话的权限

(1) 在工程中选中 res→AndroidManifest. xml 文件，如图 7-1 所示。

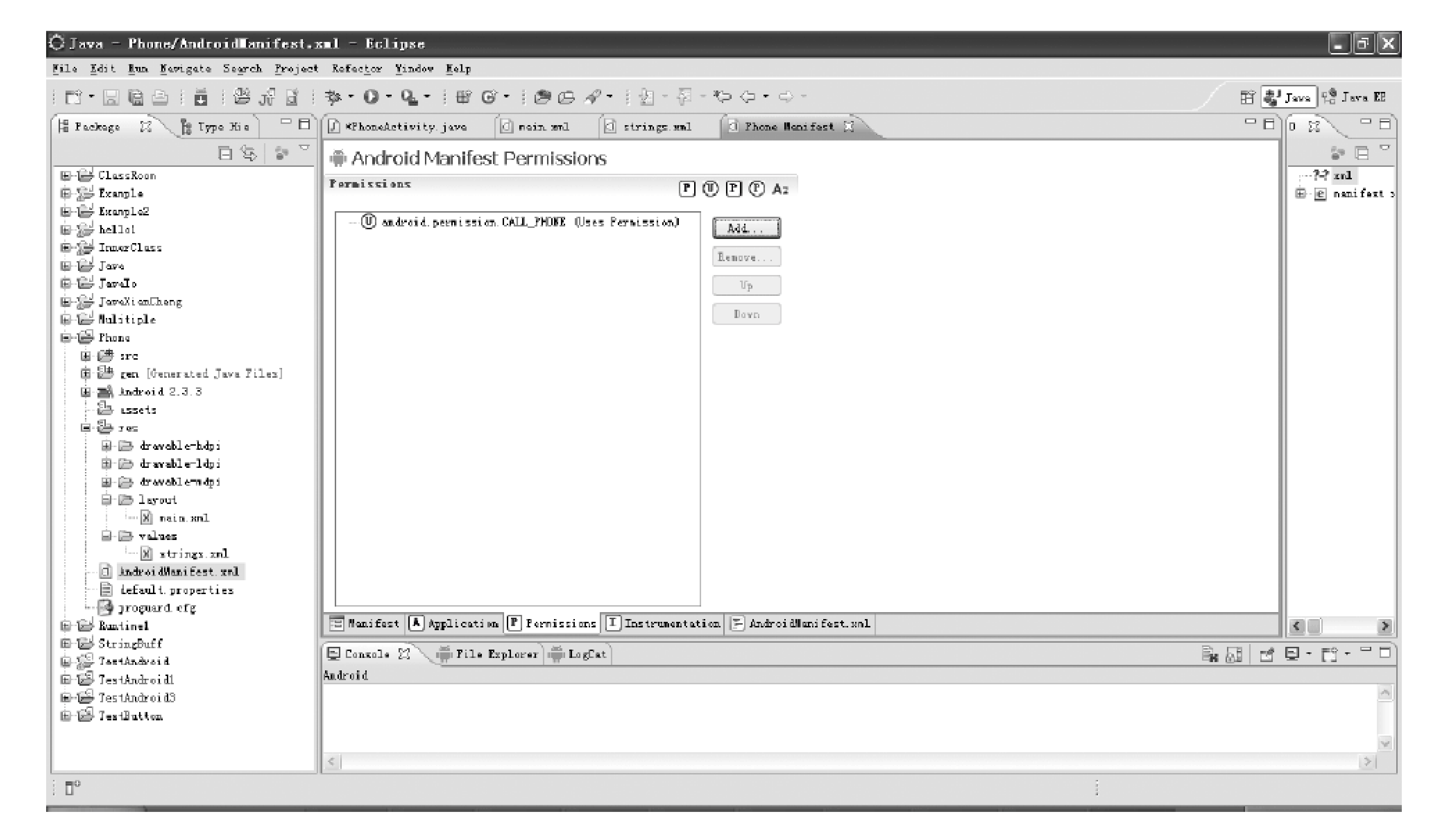


图 7-1 AndroidManifest. xml 文件

(2) 单击 Add 按钮,打开在 Manifest 中创建新元素对话框,如图 7-2 所示,在 Manifest 中创建一个新元素。

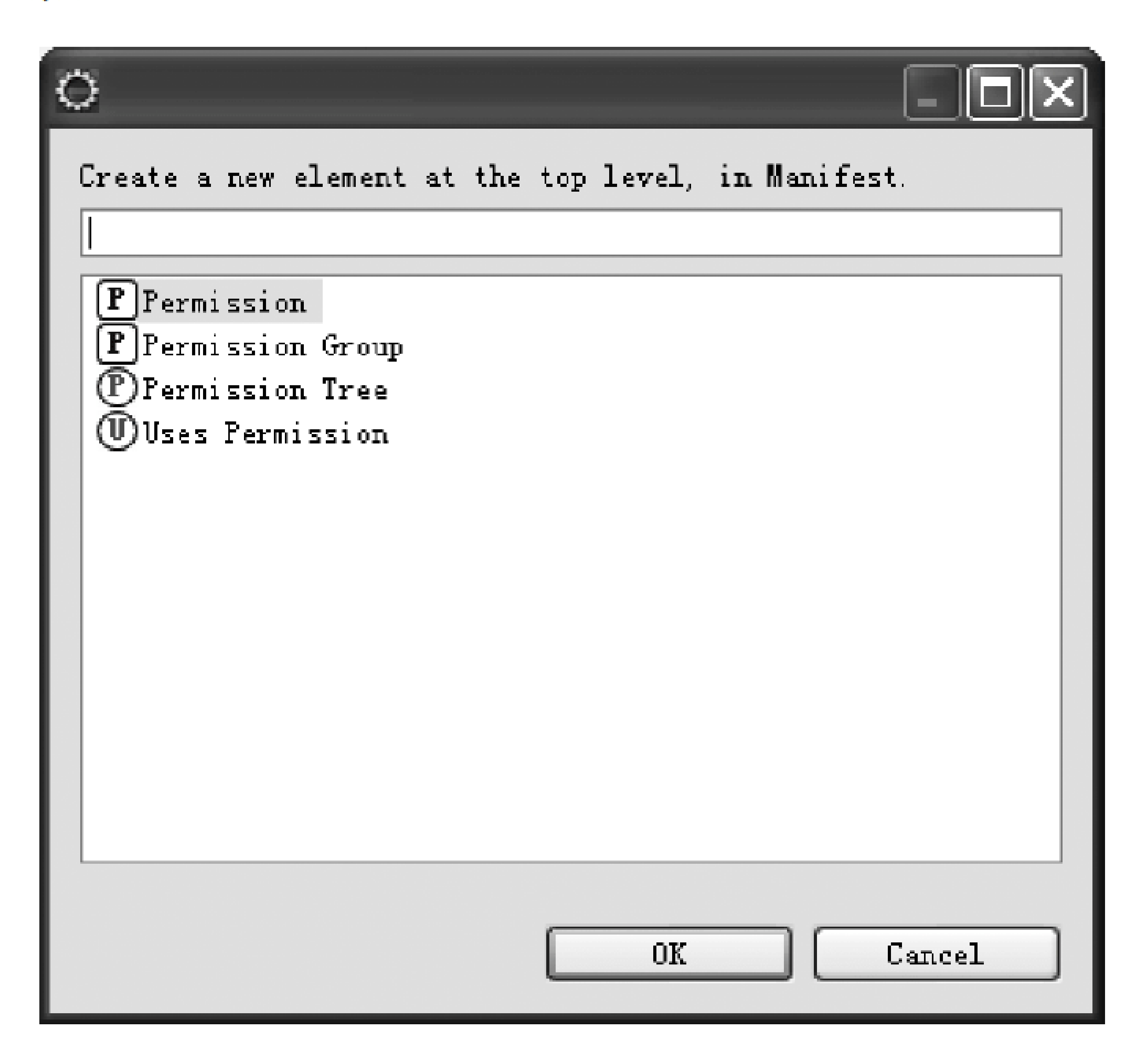


图 7-2　在 Manifest 中创建新元素对话框

(3) 单击 Uses Permission 选项,出现用户授权窗口,如图 7-3 所示。

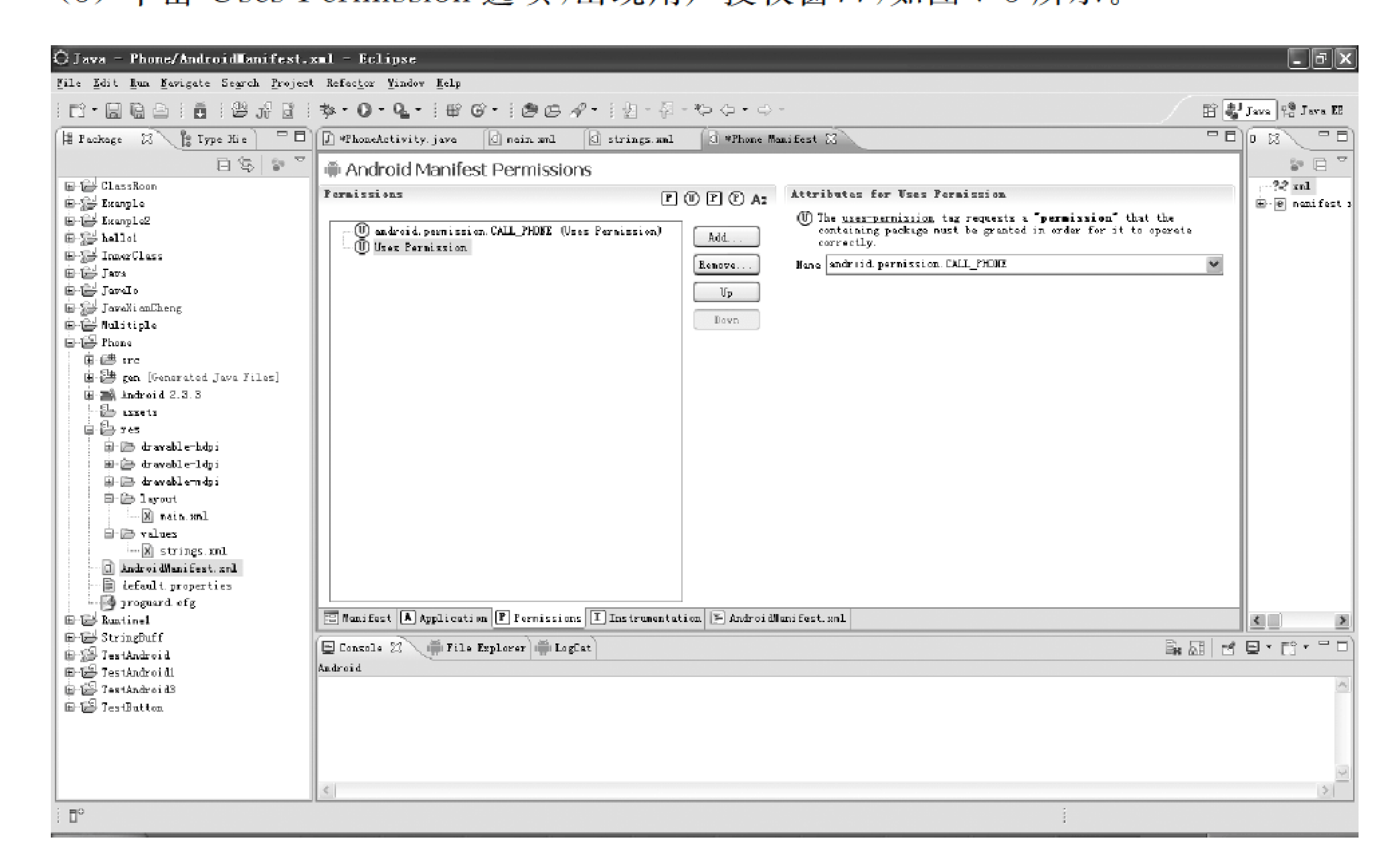


图 7-3　用户授权窗口

这样就可以运行了!

7.2 实例 2：通讯录模块的设计与实现

7.2.1 功能要求

1. 对联系人的管理

首先要完成联系人的增、删、改、查 4 个基本功能，其次能够对联系人进行分组。

(1) 在初始化联系人时有组别属性，能对联系人组别进行设置。

(2) 对已生成的联系人能够更改其组别属性，包括将联系人添加到某一新组、将联系人从某一组中移除、将联系人从 A 组中移到 B 组。

(3) 完成对联系人的批量管理、批量删除、批量转移组别。

(4) 对选定联系人打电话、发短信、发邮件。

2. 对分组的管理

要完成分组的增、删、改 3 个基本功能以及附加功能（从手机、SIM 卡上导入联系人）。

7.2.2 设计思路

在项目的整体框架搭建时，要学会应用 MVC 框架思想，即分层分包，如图 7-4 所示。

顶层是用户界面层（UI 层），主要是一些 Activity，每一个 Activity 都会被 res-res-layout 中的某个 xml 视图绑定；下一层是业务逻辑层（BIZ 层），要实现功能的业务逻辑一般都在这一层完成，它是对 DAO 层提供的方法的面向 UI 的一种封装；再下一层是数据接口层（DAO 层），DAO 层又可分为 DAO 接口抽象、DAO 方法实现，其中. dao 是 DAO 层的接口抽象，. daompl 是具体的方法实现；再下一层是基类层（MODEL 层），这一层主要放一些基类即自定义的对象。

一般主要是这 4 层，当涉及数据库操作时，还有. db、. dbconstant 层。在. db 中是一些 DAO 层涉及数据操作的方法实现，而真正的 SQL 语句是在. dbconstant 包中；在 Android 中，SQLite 数据库是动态生成的，所有. dbconstant 中还包括了数据库的生成、升级等 SQL 语句。

在进行数据库操作时，要学会使用基本的 SQL 语句 insert/delete/update/select 及 SQL 语句的拼凑，如 String. format 和 Message. format；还要注意 SQL 语句中有 3 种占位符%s、? 和{}；其他还有 test 包（其中为单元测试时写的测试类），tools 类则是一些方法的封装，当然包名是可以自定义的。

Android 工程可由 src、gen、assets、bin、res 目录和 manifest 文件等组成，如图 7-4 所

Contact
Android 2.2
src
cn.android.contact.biz
cn.android.contact.dao
cn.android.contact.daompl
cn.android.contact.db
cn.android.contact.dbconstant
cn.android.contact.model
cn.android.contact.test
cn.android.contact.tools
cn.android.contact.ui
gen [Generated Java Files]
assets
bin
res
AndroidManifest.xml
proguard.cfg
project.properties

图 7-4 项目框架图

示。其中 src 目录中存放的是项目代码文件；gen 目录中存放的是系统自动生成的一个 R 文件，不可人为地修改，里面存放着资源的索引 ID，资源包括图片、控件等；assets 目录中可以放一些自己额外需要的文件；bin 目录中存放的是编译后的.apk 文件；res 目录中主要有 drawalbe、layout、values 3 种子目录，drawalbe 中存放图片，其中又分为 3 种规格，分别对应不同的分辨率(hpi、lpi、mpi)，layout 中存放 xml 视图文件，values 中存放一些常量以便管理；最后 manifest 文件中是关于项目的一些声明，如项目名、版本号、Activity 生成声明、主 Activity 的过滤、权限的设置等都在该文件中进行设置。

7.2.3 流程图

通讯录模块的流程图如图 7-5 所示。进入程序后首先会判断是否为全屏展示，然后是一些初始化操作(如为 expandablelist 绑定数据，并设置 Item 监听)。数据的绑定会使用到继承至 BaseAdapter 的适配器，在此适配器中主要完成数据填充和按键监听(打电话、发短信、发邮件的按键监听)，其次是菜单的建立，包括底部菜单和快捷菜单。建立的过程包括注册、内容设置、单击处理；最后是一些按键的监听，监听到“新建联系人”按键和联系人条目有按下状态时，都会跳转到新建联系人界面，区别为当是联系人条目的单击事件时，新建联系人界面会得到传递过来的联系人 ID，通过 ID 去获取具体联系人信息并对应展示，而在新建联系人界面的组别设置时是通过 SelectGroupActivity 类-Dialog 样式的 Activity 进行选择的，更多属性的设置则是通过 AddMoreContactInfoActivity 类-Dialog 样式的 Activity 进行设置的。监听到“所有联系人”按键按下时，会跳转到副界面，在界面中首先会去数据库获得联系人信息并通过 listview 展示，在此界面中菜单条目、联系人条目的监听及单击事件的响应、操作与主界面中的一致，在此界面中还要实现通过联系人的姓名或手机号进行精确及模糊查询。

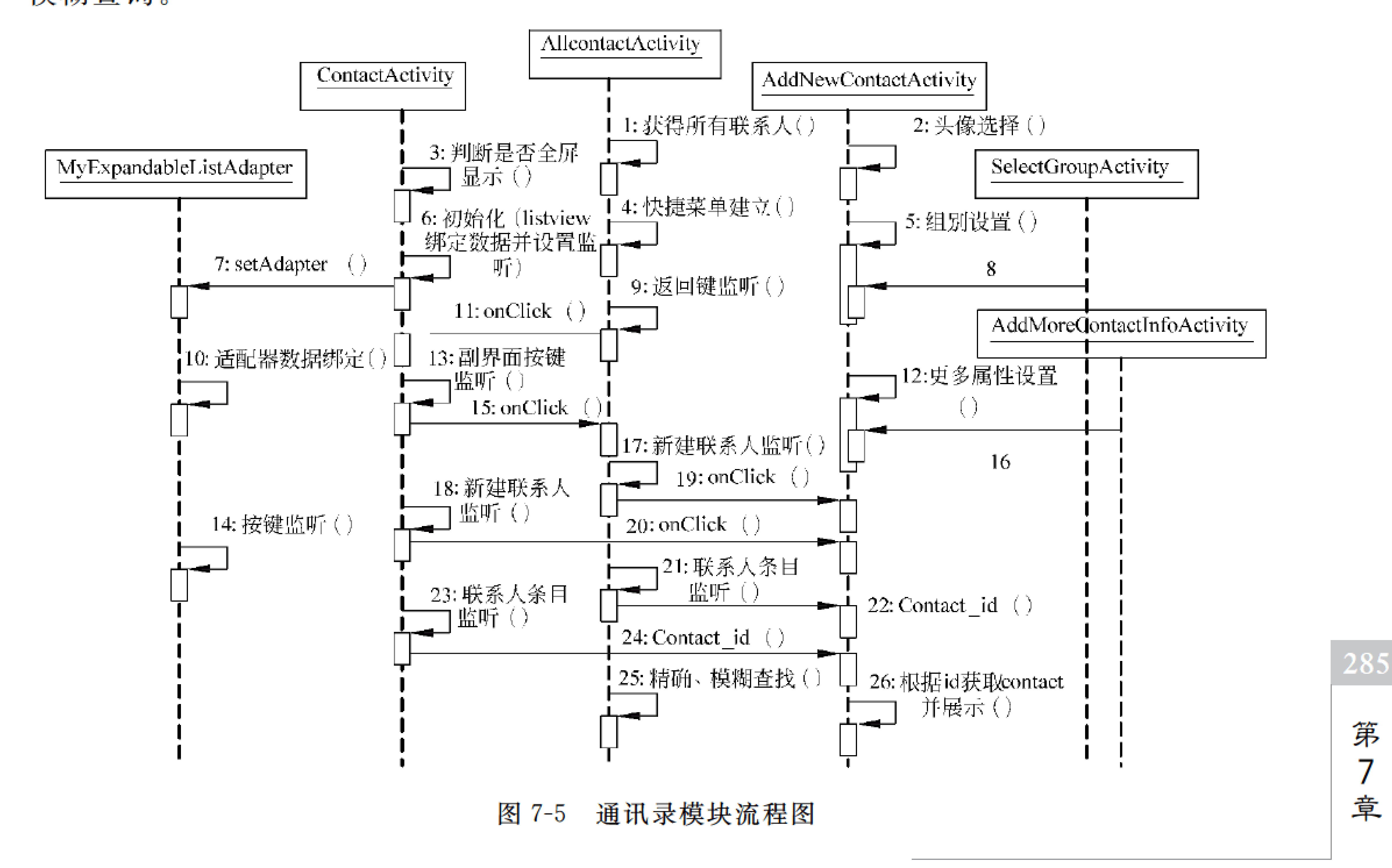


图 7-5 通讯录模块流程图

7.2.4 主界面设计与实现

1. 主界面效果图

主界面效果图如图 7-6 所示。对主界面进行设计时要注意 UI 要点和知识要点。

1）UI 要点

顶部左右为两个快捷键，左边的为切换到副界面的快捷键，右边的为切换到添加联系人界面的快捷键；中间主体部分为联系人的分组展示，组别前的小图标分为收起和展开两种状态；联系条目的右侧图标为触发 PopupWindow 弹出的按钮；底部，当为标记状态时会有标记图标显示，单击 MENU 键能够调出底部菜单。



图 7-6　主界面效果图

2）知识要点

对 Activity 生命周期的理解和应用及 Activity 样式的使用，3 种展示形式 Dialog、NoTitle 和全屏。其中全屏可使用 getWindow().setFlags 进行设置；多个 Activity 间跳转时要注意数据的传递，可以在 startActivityForResult、onActivityResult 这两个方法中进行设置；对数据进行展示时是应用 ExpandableListView，涉及数据的关联，实时刷新；注意要防止小图标变形，通过设置 setCacheColorHint(0)来防止条目单击变黑。

2. 菜单控件的使用

在使用菜单控件 contextmenu 时，要注意它的使用步骤，先注册 setOnCreateContextMenuListener，再设置 menu 内容 onCreateContextMenu，最后设置菜单条目单击事件的 onContextItemSelected。

图 7-7 所示为长按组别时弹出的快捷菜单，菜单功能为实现添加新分组、删除此分组、此分组重命名、为此分组添加新的联系人、将此分组中的联系人全部移动到其他组别和取消此快捷菜单。图 7-8 所示为普通状态（与标记状态相对）长按联系人时弹出的快捷菜单，菜单功能为实现进入标记状态、删除此联系人、修改此联系人信息、将此联系人移动到其他分组和取消此快捷菜单。图 7-9 所示为标记状态长按联系人时弹出的快捷菜单，菜单功能为实现退出标记状态、全部标记（包括其他分组）、取消所有现有标记、删除有标记状态的联系人、将有标记状态的联系人移动到指定分组内和取消此快捷菜单。



图 7-7　长按组别时的快捷菜单



图 7-8　普通状态时的快捷菜单



图 7-9　标记状态时的快捷菜单

3. 系统控件设置和 PopupWindow 的使用

在调用系统控件进行操作时要进行权限的设置，如打电话、发短信、发 E-mail。另外还有 PopupWindow 的使用步骤：先 inflate 反射出 layout，再 new 出 PopupWindow 对象，最后 show PopupWindow。其中要注意设置 setBackgroundDrawable，使单击其他地方时 PopupWindow 能够 dismiss。

其中打电话权限的主要实现代码如下：

```
Intent intent = new Intent(Intent.ACTION_CALL);
intent.setData(Uri.parse("tel:" + phone));
startActivity(intent);
```

4. 标记联系人及修改联系人信息

在标记状态下可对联系人进行标记，然后进行操作(如批量删除、批量移动等)。界面底部有“标记”字样图标提示用户当前为标记状态，效果图如图 7-10 所示。

在主界面单击条目时进入联系人详情界面，其效果图如 7-11 所示。此界面添加联系人界面的复用，只是在初始化的时候添加了指定数据。刚进入此界面为浏览界面，用户还不能对联系人信息进行修改，只有单击右上角的铅笔图标时才会进入编辑状态，进行修改联系人信息的操作。左上侧的图标为返回键。

5. 底部菜单

在主界面的底部菜单中单击“全屏”命令可以进行个性化设置，如图 7-12 所示。单击“全屏”命令后确定，此次运行软件全屏，下次再运行恢复非全屏设置；单击“全屏”命令，再选择“记住我”，后确定，此次运行软件全屏，下次再运行保持全屏状态。



图 7-10 标记状态下的主界面图



图 7-11 联系人详情界面



图 7-12 底部菜单单击“全屏”命令后

在主界面的底部菜单中单击“工具箱”命令后，跳转到工具箱界面，效果图如图 7-13 所示。目前只实现了第一项和第二项两个功能，分别为从手机、SIM 中将联系人导入到通讯录中，后续完善项目时可以继续完成其他功能，也可以将新的功能添加到工具箱内。

7.2.5 副界面设计与实现

(1) 副界面设计时的 UI 要点：主要是多了一个搜索框，用户在搜索框内输入关键字后能快速地定位要特定的联系人。

(2) 知识要点：除了联系人展示方式不同并多了搜索功能外，其他功能与主界面的相同，此界面中的展示功能利用 listView，搜索功能则是利用 SQL 语句的拼装来实现实时查找的。

(3) 副界面的效果图如图 7-14 所示。



图 7-13 工具箱界面



图 7-14 副界面效果图

7.2.6 添加联系人界面设计与实现

在添加联系人界面中，人物头像的选择可以从 3 个方面进行获取：拍照、系统图库及读文件；出生日期的选择涉及对 DatePicker 控件的使用。效果图如图 7-15 所示。

(1) 单击头像图标时，利用 PopupWindow 弹出 3 个选项供用户选择，效果图如图 7-16 所示。“拍照”是调用系统相机拍照，将拍好的图片获取过来作为联系人头像；“系统图片”是项目配置好的静态图片，利用 imageswitcher 和 gallery 进行展示；“从文件选择”即从 SD 中获取图片资源，主要涉及文件流操作。



图 7-15 添加联系人界面效果图



图 7-16 单击头像图标后界面效果图

(2) 单击“分组”按钮后，弹出 Dialog 样式的 Activity 供用户进行单项选择，效果图如图 7-17 所示。

(3) 单击“添加更多属性”按钮后，弹出 Dialog 样式的 Activity 供用户进行单项选择，效果图如图 7-18 所示。此界面为图 7-17 所示的“选择分组”界面的复用。当用户选择某项后，用户在此次添加联系人操作过程中再次调出此选择界面时不再出现先前选择过的条目。例如用户当前选择了“家庭电话”条目，则当用户再次单击“添加更多属性”按钮时，“请选择类型”选项表中不再有“家庭电话”项。




图 7-17　单击“分组”按钮后



图 7-18　单击“添加更多属性”按钮后

(4) 其他方面，如联系人的导入，是针对需要在系统数据库中查找对应字段及内容提供者的信息。内容提供者的 URL 是 content:/icc/and。在对联系人进行获取时要进行读取联系人权限设置，代码如下：

```
<uses-permission android:name = "android.permission.READ_CONTACTS"/>
<uses-permission android:name = "android.permission.CALL_PHONE"/>
```

7.2.7　调试

调试方面，可以设置权限进行单元测试，Debug 或者是看 Log 输出。单元测试在 mainifest 内的权限设置如下：

```
<instrumentation
     android:label = "Tests for My App"
     android:name = "android.test.InstrumentationTestRunner"
     android:targetPackage = "cn.android.contact" />
```

单元测试要独包独类，把需要测试的方法一个个地检验过去。

7.2.8　通讯录模块功能实现代码

```
    public class ContactActivity extends Activity {
    public final static int GROUP_TO_ADD_CONTACT = 0;// 由分组菜单项跳转到新建联系人界面
    public final static int CTREAT_TO_ADDNEW = 1;
                                           // 单击右上角"新建"图标跳转到新建联系人界面
```

```
public final static int GROUPCONTACT_MOVETO = 2;  // 全组成员移动跳转到组别选择
public final static int CONTACT_MOVETO = 4;       // 单人移动跳转到组别选择
public final static int EDIT_CONTACT = 11;        // 单击"编辑"跳转到修改联系人界面
public final static int MARK_MOVE_TO = 5;         // 单击标记移动..跳转到组别选择
public final static int INFO = 6;                 // 单击查看详情
private ExpandableListAdapter adapter;
private ExpandableListView expandableListView;
private Context context;
private GroupManager groupManager;
private ContactManager contactManager;
private ImageView imgContacts;                   // 左上角切换到联系人列表的 ImageView
private ImageView imgCreate;                     // 右上角切换到新建联系人的 ImageView
private RelativeLayout rl_mark;
private boolean[] isOpen;                  // 用于处理 expandableListView 箭头小图标拉伸
private int moveToGroupId = -1;                   // 移动操作的目标分组 ID
private int moveReGroupId;                        // 移动操作的原分组 ID
private int moveReContactId;                      // 移动操作的联系人 ID
private boolean isMarkedState = false;            // 是否是标记状态
private boolean isFirst = true;           // 为了第一次进入 onResume 而不执行其内的操作
private List<Group> list;                         // 存放所有的 group
private GridView bottomMenuGrid;                  // 屏幕下方主菜单的布局
private String[] bottom_menu_itemName = { "工具箱", "全屏", "退出" };    // 主菜单文字
private int[] bottom_menu_itemSource = { R.drawable.menu_tool,          // 主菜单图片
        R.drawable.full_screen, R.drawable.menu_exit };
// 为 MyExpandableListAdapter 建立的数据源
private List<String> groupItem = new ArrayList<String>();     // 存放所有的组别名称
private List<List<Contact>> contacts = new ArrayList<List<Contact>>();
                                                          // 按组别存放 contact
private List<Integer> groupCount = new ArrayList<Integer>();
                                                  // 存放各分组内 contact 的数量
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    isMakeFull();
    setContentView(R.layout.main);
    context = this;
    groupManager = new GroupManager(context);
    contactManager = new ContactManager(context);
    // 初始化标记状态,false:全部未标记; true:全部标记
    contactManager.initIsMark("false");
    getId();
    expandableListView.setGroupIndicator(null);   // 箭头小图标,不用自带的了
    expandableListView.setCacheColorHint(0);      // 防止 expandableListView 单击变黑
    // 为 expandableListView 注册上下文菜单 SETP.1
    expandableListView.setOnCreateContextMenuListener(this);
    // 绑定数据,刷新界面
    bindView();
    // 头像选择监听
    // Group 单击监听
    expandableListView.setOnGroupClickListener(new OnGroupClickListener() {
```

```
            public boolean onGroupClick(ExpandableListView mExpandableList,View arg1, final
int groupPosition, long id) {
                return false;
            }
        });
        // Contact 单击监听
        expandableListView.setOnChildClickListener(new OnChildClickListener() {
                public boolean onChildClick (ExpandableListView parent, View v, int
groupPosition, int childPosition, long id) {
                Contact contact = contacts.get(groupPosition)
                        .get(childPosition);
                if (isMarkedState) {
                    ImageView img = (ImageView)v.findViewById(R.id.imgChk);
                    if (img.getVisibility() == View.GONE) {
                        contactManager.initIsMarkById("true", contact.getId());
                        img.setVisibility(View.VISIBLE);
                    } else {
                        contactManager.initIsMarkById("false", contact.getId());
                        img.setVisibility(View.GONE);
                    }
                } else {
                    Intent intent = new Intent(context,
                            AddNewContactActivity.class);
                    Bundle bundle = new Bundle();
                    bundle.putInt("requestCode", INFO);
                    bundle.putInt("id", contact.getId());
                    intent.putExtras(bundle);
                    startActivity(intent);
                }
                return false;
            }
        });
        // 处理 expandableListView 自定义箭头小图标被拉伸 SETP.1
        expandableListView
                .setOnGroupCollapseListener(new ExpandableListView.OnGroupCollapseListener() {
                    public void onGroupCollapse(int groupPosition) {
                        isOpen[groupPosition] = false;
                    }
                });
        // SETP.2
        expandableListView
                .setOnGroupExpandListener(new ExpandableListView.OnGroupExpandListener() {
                    public void onGroupExpand(int groupPosition) {
                        isOpen[groupPosition] = true;
                    }
                });
        // 切换到联系人视图
        imgContacts.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                        Intent intent = new Intent(context,
```

```
                    AllContactShowActivity.class);
                startActivity(intent);
            }
        });
        // 新建联系人
        imgCreate.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                Intent intent2 = new Intent(context,
                        AddNewContactActivity.class);
                intent2.putExtra("requestCode", CTREAT_TO_ADDNEW);
                startActivity(intent2);
            }
        });
    }

    private void isMakeFull() {
        SharedPreferences sp = getSharedPreferences("parameter", Context.MODE_PRIVATE);
        boolean isFull = sp.getBoolean("isFull", false);
        if(isFull){
                getWindow ( ). setFlags ( WindowManager. LayoutParams.  FLAG _ FULLSCREEN,
WindowManager.LayoutParams. FLAG_FULLSCREEN);

        }
    }
    private void getId() {
        rl_mark = (RelativeLayout) findViewById(R.id.rl_mark);
        imgContacts = (ImageView) findViewById(R.id.imgContacts);
        imgCreate = (ImageView) findViewById(R.id.imgCreate);
        expandableListView = (ExpandableListView) findViewById(R.id.expandableListView);
    }
    // 调用系统发送 E-mail
    public void sendEmail(String Email) {
        Uri uri = Uri.parse("mailto:" + Email);
        Intent it = new Intent(Intent.ACTION_SENDTO, uri);
        startActivity(it);
    }
    // 调用系统发短信
    public void sendMsg(String phone) {
        if (phone.equals("")) {
            Toasttool.MyToast(context, "没有号码!");
        } else {
            Uri uri = Uri.parse("smsto:" + phone);
            Intent it = new Intent(Intent.ACTION_SENDTO, uri);
            it.putExtra("sms_body", "");
            startActivity(it);
        }
    }
    // 调用系统打电话
    public void makeCall(String phone) {
```

```
            if (phone.equals("")) {
                Toasttool.MyToast(context, "没有号码!");
            } else {
                Intent intent = new Intent(Intent.ACTION_CALL);
                intent.setData(Uri.parse("tel:" + phone));
                startActivity(intent);
            }
        }
        private void bindView() {
            getData();
            if (contacts != null) {
                if(isFirst){
                      adapter = new MyExpandableListAdapter(groupCount, groupItem, contacts,
    isOpen, context);
                    expandableListView.setAdapter(adapter);
                }else{
                    ((MyExpandableListAdapter) adapter).fresh(groupCount, groupItem,contacts,
    isOpen);
                }
            }
        }
        private void getData() {
            groupItem.clear();
            contacts.clear();
            groupCount.clear();
            list = groupManager.getAllGroup();
                    for (Group p : list) {
                groupItem.add ( p. getGroupName ( )); groupCount. add ( contactManager.
    getCountByGroupId(p. getGroupId ( ))); contacts. add ( contactManager. getContactByGroupId ( p.
    getGroupId()));
            }
            isOpen = new boolean[groupManager.getGroupCount()];
        }
        // 创建上下文菜单内容 STEP.2
        @Override
        public void onCreateContextMenu(ContextMenu menu, View v,
                ContextMenuInfo menuInfo) {
            ExpandableListView.ExpandableListContextMenuInfo info =
    (ExpandableListView.ExpandableListContextMenuInfo) menuInfo;
            int type = ExpandableListView
                    .getPackedPositionType(info.packedPosition);
            int groupPosition = ExpandableListView
                    .getPackedPositionGroup(info.packedPosition);
            if (type == 0)                              // 单击的是 GROUP
            {
                if (groupPosition != 0) {               // 默认分组禁止删除
                    menu.add(0, 1, 1, "删除分组");
                }
                // 添加菜单项
                menu.add(0, 0, 0, "添加分组");
```

```
                menu.add(0, 2, 2, "重命名");
                menu.add(0, 3, 3, "添加联系人");
                menu.add(0, 4, 5, "取消");
                menu.add(0, 5, 4, "全组成员移动..");
            } else if (type == 1) {                     // 单击的是联系人
                if (isMarkedState) {                    // 标记状态菜单
                    menu.add(2, 20, 0, "退出标记状态");
                    menu.add(2, 21, 1, "全部标记");
                    menu.add(2, 22, 2, "取消所有标记");
                    menu.add(2, 23, 3, "删除标记");
                    menu.add(2, 24, 4, "移动标记到..");
                    menu.add(2, 25, 5, "取消");
                } else {                                // 普通状态菜单
                    menu.add(1, 10, 0, "进入标记状态");
                    menu.add(1, 12, 2, "删除");
                    menu.add(1, 13, 4, "移动到..");
                    menu.add(1, 14, 5, "取消");
                    menu.add(1, 15, 3, "编辑");
                }
            }
            super.onCreateContextMenu(menu, v, menuInfo);
        }
        // menuItem 单击事件响应 STEP.3
        @Override
        public boolean onContextItemSelected(MenuItem item) {
            ExpandableListView.ExpandableListContextMenuInfo menuInfo =
    (ExpandableListView.ExpandableListContextMenuInfo) item
                    .getMenuInfo();
            int groupPosition = ExpandableListView
                    .getPackedPositionGroup(menuInfo.packedPosition);
            int childPosition = ExpandableListView
                    .getPackedPositionChild(menuInfo.packedPosition);
            Contact contact = null;
            if (childPosition != -1) {                  // 有 child 项显示再得到 contact
                contact = contacts.get(groupPosition).get(childPosition);
            }
            Group group = list.get(groupPosition);
            switch (item.getItemId()) {
            case 0:                     // 添加分组
                Intent intent = new Intent(context, AddNewGroupActivity.class);
                startActivity(intent);
                break;
            case 1:                     // 删除分组
                contactManager.changeGroupByGroup(group.getGroupId(), 1);
                                                        // 将组成员移动到默认分组中
                groupManager.deleteGroupByID(group.getGroupId());          // 删除指定的分组
                bindView();
                break;
            case 2:                     // 重命名
                Intent intent1 = new Intent(context, AddNewGroupActivity.class);
```

```java
            Bundle bundle = new Bundle();
            bundle.putInt("groupId", group.getGroupId());
            bundle.putString("groupName", group.getGroupName());
            intent1.putExtras(bundle);
            startActivity(intent1);
            break;
        case 3:                         // 添加联系人
            Intent intent2 = new Intent(context, AddNewContactActivity.class);
            Bundle bundle1 = new Bundle();
            bundle1.putInt("groupId", group.getGroupId());
            bundle1.putString("groupName", group.getGroupName());
            bundle1.putInt("requestCode", GROUP_TO_ADD_CONTACT);
            intent2.putExtras(bundle1);
            startActivity(intent2);
            break;
        case 4:                         // 取消
            break;
        case 5:                         // 全组成员移动..
            moveReGroupId = group.getGroupId();
            Intent intent3 = new Intent(context, SelectGroupActivity.class);
            intent3.putExtra("requestCode", GROUPCONTACT_MOVETO);
            startActivityForResult(intent3, 0);
            break;
        case 10:                        // 进入标记状态
            rl_mark.setVisibility(View.VISIBLE);
            isMarkedState = true;
            break;
        case 12:                        // 删除
            if (contact != null)
                contactManager.deleteById(contact.getId());
            bindView();
            break;
        case 13:                        // 移动到..
            Intent intent4 = new Intent(context, SelectGroupActivity.class);
            intent4.putExtra("requestCode", CONTACT_MOVETO);
            startActivityForResult(intent4, 0);
            if (contact != null)
                moveReContactId = contact.getId();
            break;
        case 14:                        // 取消
            break;
        case 15:                        // 编辑
            Intent intent5 = new Intent(context, AddNewContactActivity.class);
            Bundle bundle2 = new Bundle();
            bundle2.putInt("id", contact.getId());
            bundle2.putInt("requestCode", EDIT_CONTACT);
            intent5.putExtras(bundle2);
            startActivity(intent5);
            break;
        case 20:                        // 退出标记状态
```

```
                rl_mark.setVisibility(View.GONE);
                contactManager.initIsMark("false");
                isMarkedState = false;
                ((MyExpandableListAdapter) adapter).refresh();
                break;
            case 21:                        // 全部标记
                contactManager.initIsMark("true");
                // bindView();
                ((MyExpandableListAdapter) adapter).refresh();
                break;
            case 22:                        // 取消所有标记
                contactManager.initIsMark("false");
                ((MyExpandableListAdapter) adapter).refresh();
                break;
            case 23:                        // 删除标记
                contactManager.deletByIsMark();
                bindView();
                break;
            case 24:                        // 移动标记到..
                 Intent intent6 = new Intent(context, SelectGroupActivity.class);
intent6.putExtra("requestCode", MARK_MOVE_TO);
                startActivityForResult(intent6, 0);
                break;
            case 25:                        // 取消
                break;
            default:
                break;
            }
            return super.onContextItemSelected(item);
        }
        @Override
        protected void onResume() {
            if (!isFirst) {
                bindView();
            }
            isFirst = false;
            super.onResume();
        }
        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
                if (resultCode == 2 && data != null) {
                moveToGroupId = data.getExtras().getInt("groupId");
                contactManager.changeGroupByGroup(moveReGroupId, moveToGroupId);
            } else if (resultCode == 4) {
                moveToGroupId = data.getExtras().getInt("groupId");
                contactManager.changeGroupByCotact(moveReContactId, moveToGroupId);
            } else if (resultCode == 5) {
                moveToGroupId = data.getExtras().getInt("groupId");
                contactManager.changeGroupByMark(moveToGroupId);
                contactManager.initIsMark("false");
```

```
        } else if (resultCode == 500) {
            List<Object> list = (List<Object>) data.getExtras().get("contacts");
            List<String> name = (List<String>) list.get(0);
            List<String> phone = (List<String>) list.get(1);
            List<Bitmap> photo = (List<Bitmap>) list.get(2);
            for (int i = 0; i < name.size() ; i++) {
                Contact contact = new Contact(
                        ImageTools.getByteFromBitmap(photo.get(i)),
                        name.get(i), 1, phone.get(i), "", "", "", "",
                        "", "", "false");
                contactManager.addContact(contact);
                Log.i("Item", contact.toString());
            }
        } else if (resultCode == 600) {
        }else if(resultCode == 1000){
            getWindow(). setFlags ( WindowManager. LayoutParams. FLAG _ FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        super.onActivityResult(requestCode, resultCode, data);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_MENU) {
            loadBottomMenu();
            if (bottomMenuGrid.getVisibility() == View.VISIBLE) {
                bottomMenuGrid.setVisibility(View.GONE);
            } else {
                bottomMenuGrid.setVisibility(View.VISIBLE);
            }
        }
        return super.onKeyDown(keyCode, event);
    }
    // button_menu 主菜单的 Adapter
    private SimpleAdapter getMenuAdapter(String[] menuNameArray,
            int[] imageResourceArray) {
        ArrayList<HashMap<String, Object>> data = new ArrayList<HashMap<String, Object>>();
        for (int i = 0; i < menuNameArray.length; i++) {
            HashMap<String, Object> map = new HashMap<String, Object>();
            map.put("itemImage", imageResourceArray[i]);
            map.put("itemText", menuNameArray[i]);
            data.add(map);
        }
        SimpleAdapter simperAdapter = new SimpleAdapter(this, data,
                R.layout.item_menu, new String[] { "itemImage", "itemText" },
                new int[] { R.id.item_image, R.id.item_text });
        return simperAdapter;
    }
    private void loadBottomMenu() {
        if (bottomMenuGrid == null) {
                bottomMenuGrid = (GridView) findViewById ( R. id. gv _ buttom _ menu );
bottomMenuGrid.setBackgroundResource(R.drawable.channelgallery_bg);  // 设置背景
```

```
            bottomMenuGrid.setNumColumns(3);                    // 设置每行列数
            bottomMenuGrid.setGravity(Gravity.CENTER);          // 位置居中
            bottomMenuGrid.setVerticalSpacing(10);              // 垂直间隔
            bottomMenuGrid.setHorizontalSpacing(10);            // 水平间隔
        bottomMenuGrid.setAdapter(getMenuAdapter(bottom_menu_itemName,
                bottom_menu_itemSource));                       // 设置菜单 Adapter
            /** 监听底部菜单选项 **/
            bottomMenuGrid.setOnItemClickListener(new OnItemClickListener() {
                public void onItemClick(AdapterView<?> arg0, View arg1,
                        int item, long arg3) {
                    switch (item) {
                    case 0:                                     // 工具箱
                        //Toasttool.MyToast(context, "工具箱");
                        Intent intent = new Intent(context,ToolGriewActivity.class);
                        startActivityForResult(intent, 8);
                        break;
                    case 1:                                     // 全屏
                        Intent intent2 = new Intent(context,FullScreenActivity.class);
                        startActivityForResult(intent2, 6);
                        break;
                    case 2:                                     // 退出
                        finish();
                        break;
                    default:
                        break;
                    }
                }
            });
        }
    }
}
```

7.3 实例 3：短信模块的设计与实现

7.3.1 功能要求

（1）要满足同时向多人发送短信的需求，实现短信群发功能。

（2）快速查寻所需信息，实现短信查找功能。

（3）为了满足特殊需求，能实现定时发送指定短信。

（4）保护个人信息安全，能给需要的短信加密。

（5）对数据的转移，实现短信的导入/导出功能，将信息内容导出成文本，反之将文本内容导入为短信。

（6）对重要数据的处理，实现短信备份功能，备份到本地或网络的另一端。

7.3.2 设计思路

在程序第一次运行时会注册一个广播用来收听是否有新短信到来，如果有新短信到来则在通知栏进行展示，且此时若软件正在运行，处于主界面则刷新主界面，处于会话列表界面且新短信归属于此会话，则刷新会话列表界面，在其他情况则刷新主界面数据；若软件未运行，则可以通过通知栏跳转，一人单条或多条短信时直接跳转到会话列表界面，多人多条短信时则跳转到主界面。

在主界面中，首先要管理通知栏、通知栏状态的清除，接着 bindlistview（从系统数据库中获取所有会话，并填充到 listview 的适配器中），然后是快捷菜单的建立、短信接收广播的建立，最后是 refresh 操作，包括新来短信的界面刷新以及会话列表返回草稿的展示。

在会话列表界面，首先初始化界面数据，判断最新一条短信类型，若为草稿，则会话内容展示的适配器的条目个数应减 1，并将草稿展示在输入框内；接着，如果此会话联系人存在多个联系号码时，对多个号码手势切换更改号码显示的监听；然后为发送短信注册内部类广播，当短信发送成功后及时更新发送短信的状态。

7.3.3 流程图

根据以上设计思路，短信模块的流程图如图 7-19 所示。

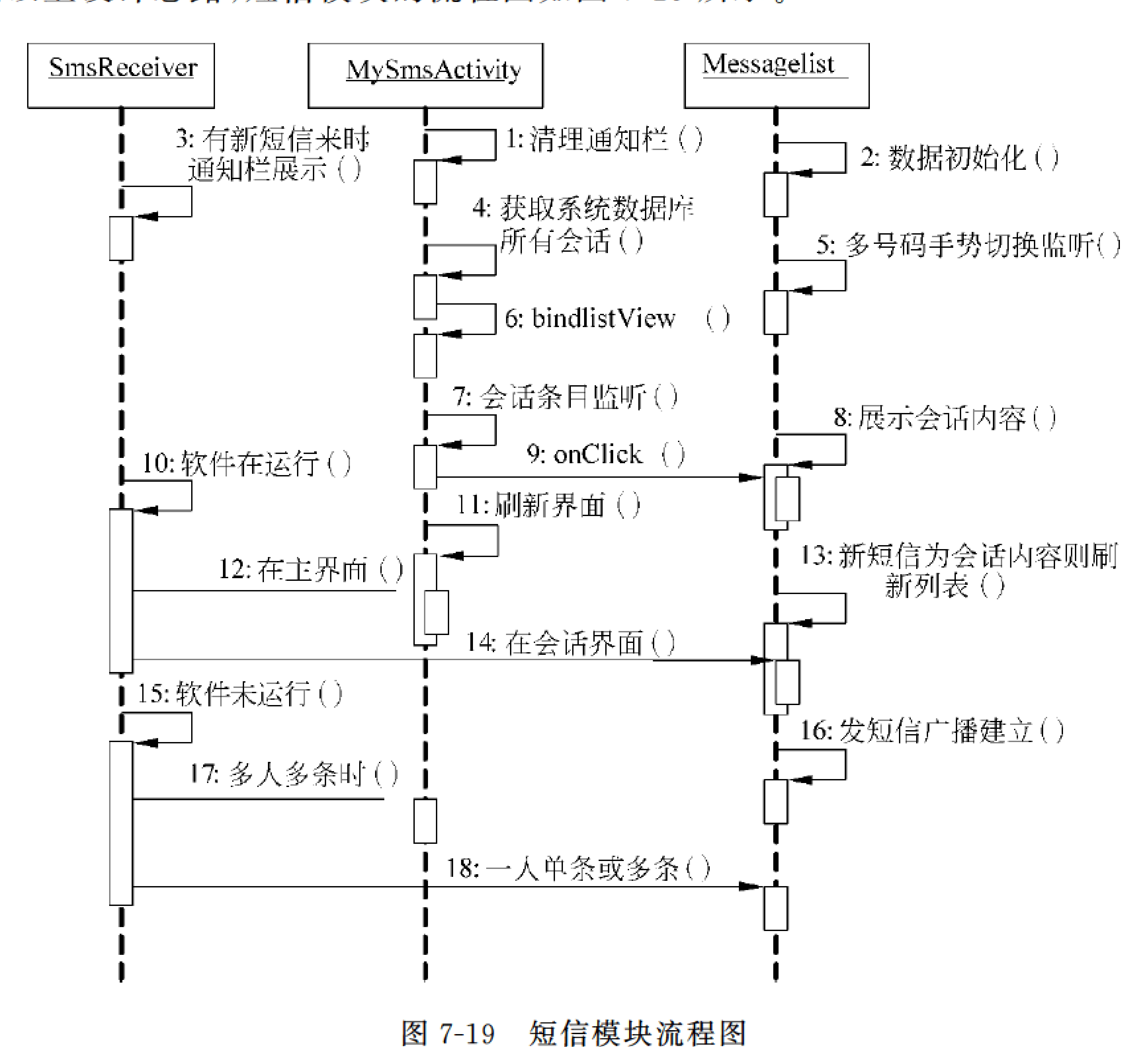


图 7-19 短信模块流程图

7.3.4 主界面设计与实现

主界面的设计主要是布局及数据的获取。数据的获取主用利用 SQL 语句，如去重“distinct”，注释同行之后的内容“——”。其中的注意点是，系统数据库中的 person 字段当联系人存在系统的拨号中时有值，存在系统的通讯录中为 null。草稿的提取展示时要注意，系统默认存入 sms 表中无号码。关于数据的实时刷新，主要是应用广播机制，要注意广播的注册与取消。

1. 系统数据库

(1) sms 表：包括所有短信。

(2) threads 表：不提供接口，通过 uri 不能直接定位到此表。

(3) canonical_address 表：号码在此表中，_id 在 threads 表中。

(4) 获取草稿号码的方法：在 sms 表中获得 thread_id 字段，根据 thread_id 在 threads 表中得到对应的 recipient_idss，最后通过 recipient_idss 在 canoniacal_address 表中获得 address，即会话号码。示意图如图 7-20 所示。

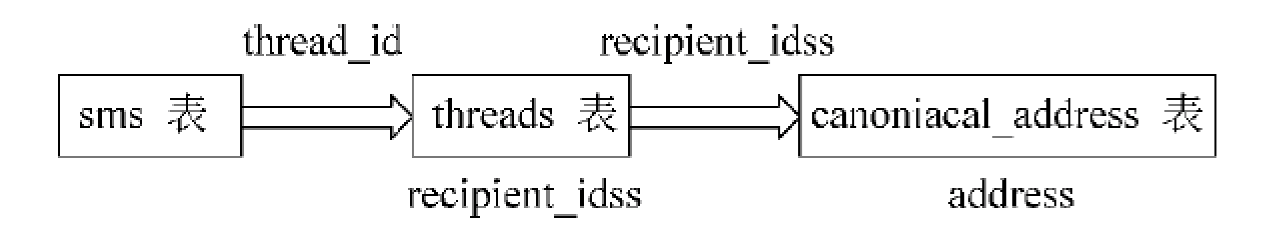


图 7-20　获取草稿号码示意图

2. 主界面

主界面的整体效果图如图 7-21 所示。

主界面要分条目展示各联系人的短信情况，具体条目的设计如图 7-22 所示。



图 7-21　主界面效果图

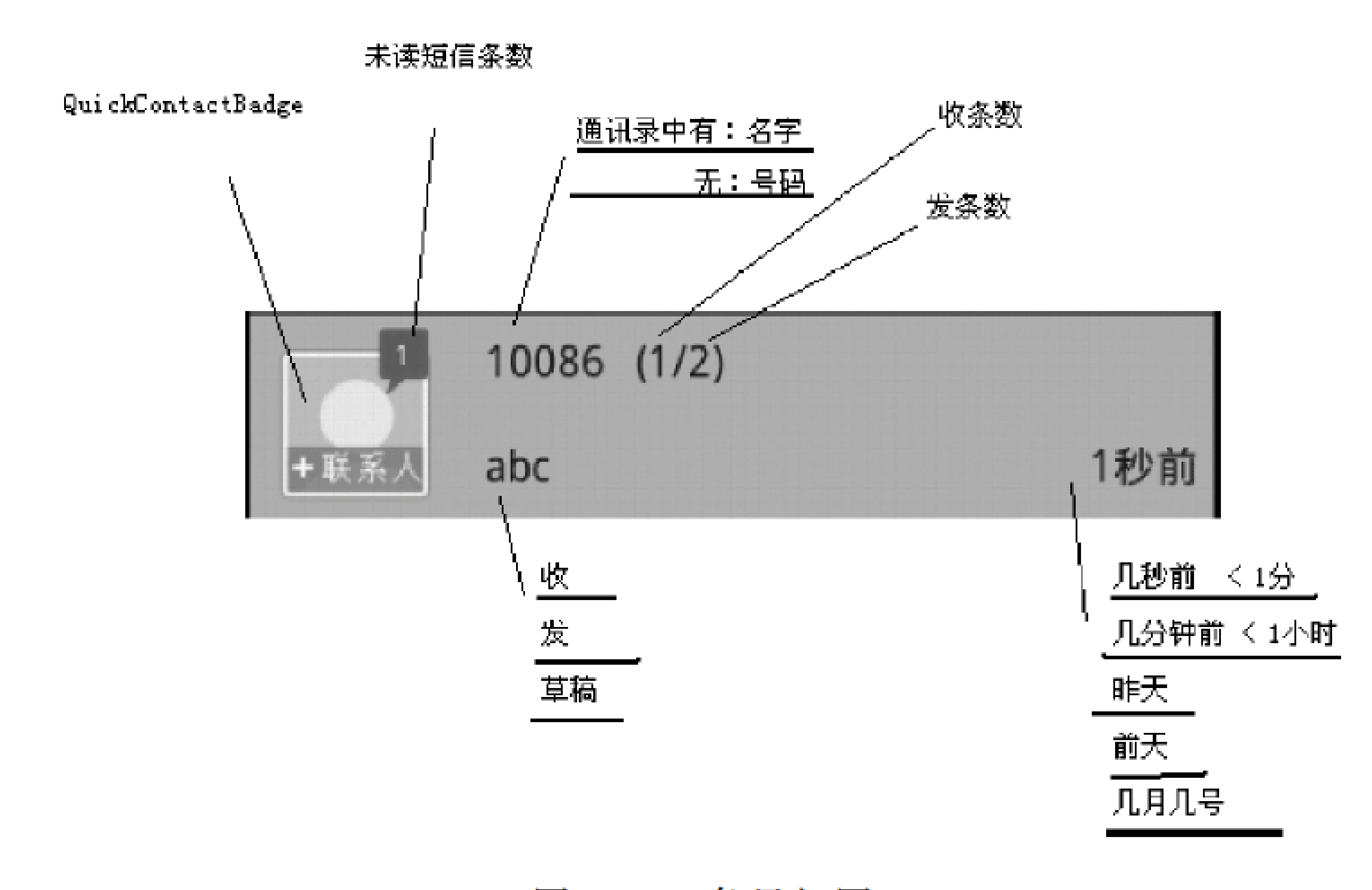


图 7-22　条目细图

7.3.5 会话列表界面设计与实现

设计会话列表界面时，主体内容的展示主要应用 ViewStub 和自定义控件，ViewStub 是为了分散代码，自定义控件是为了实现个性化功能。其中复制/粘贴功能利用了

ClipboardManager 类；头部联系人存在多个号码，可以通过手势切换，主要利用 ViewFlipper 控件。输入内容的图文输入可以利用 Html.fromHtml(source,imageGetter,tagHandler)方法对图文进行编译。

(1) 会话列表界面要实现的功能：收、发短信的分列展示；短信内容的自定义复制；发送短信的状态实时更新(从发送中到已发送)；当某个联系人存在多个号码时，可以通过手势切换不同的号码，同时短信内容变为对应号码下的内容；输入内容实现图、文输入。

(2) 界面效果图如图 7-23 所示。

(3) 短信发送中，显示文字“发送中…”；发送成功后，通过广播机制实时更新文字为“已发送”，效果图如图 7-24 所示。



图 7-23　会话列表界面效果图



图 7-24　短信发送效果图

(4) 短信内容可自定义复制/粘贴。其中 ClipboardManager 类的使用方法如下：

实例化：

```
getSystemService(Context.CLIPBOARD_SERVICE)
```

复制：

```
clip.setText();
```

粘贴：

```
clip.getText();
```

光标选择方法如下：

获取光标所在位置：

```
EditText.getSelectionStart();
```

设置光标选中：

```
Selection.setSelection(text,index);
Selection.setSelection(text,start,stop);
```


复制短信内容效果图如图 7-25 所示。

(5) 头部的显示。如果联系人存在多个号码则可以通过手势切换号码，同时会话内容切换为对应号码下的会话内容。头部效果图如图 7-26 所示。



图 7-25 复制短信内容效果图



图 7-26 会话列表头部效果图

7.3.6 通知栏设计与实现

通知栏的设计主要考虑一人单条、一人多条和多人多条这 3 种情况的不同展示，示例图分别如图 7-27、图 7-28 和图 7-29 所示，利用 SharedPreferences 数据保存，进行 3 种情况的区分判断。单击清除时，可以打开服务进行对应的操作；单击查看时，利用 pendingIntent 类的 getActivity(context,requestCode,intent,flags)方法进行配置，其中 flags 参数设置为 PendingIntent.FLAG_UPDATE_CURRENT 时数据才能实时更新。

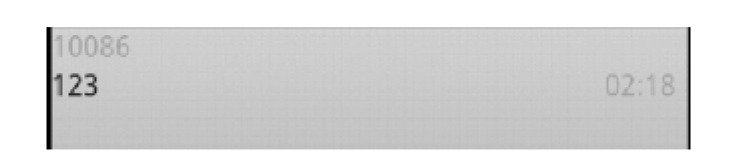


图 7-27 一人一条情况示例图

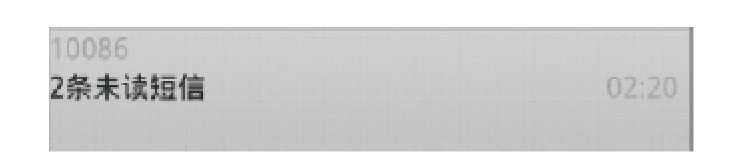


图 7-28 一人多条情况示例图



图 7-29 多人多条情况示例图

7.3.7 短信模块功能实现代码

```
public class MySMSActivity extends Activity {
private Context context;
private ImageView imgNewSMS;// 新建短信按键
private MainListAdapter adapter;
private ListView listView;
private MessageManager messageManager;
private ContactManager contactManager;
private boolean isFirst = true;
private List<Message> list;
private String broadcastPhone;
```

```
private boolean isFromBroadcast = true;
private int thread_id;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 获取通知管理器,用于发送通知
    NotificationManager manager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);
    Notification notification = new Notification();
    manager.notify(1, notification);
    SharedPreferences sp = getSharedPreferences("Message",
            Context.MODE_PRIVATE);
    Editor editor = sp.edit();
    editor.putBoolean("isFirst", true);
    editor.putInt("count", 0);
    editor.putString("strFrom", null);
    editor.commit();
    setContentView(R.layout.main);
    context = this;
    messageManager = new MessageManager(context);
    contactManager = new ContactManager(context);
    getId();
    bindListView();
    // 为 ListView 注册上下文菜单 SETP.1
    listView.setOnCreateContextMenuListener(this);
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> arg0, View v, int arg2,
                long id) {
            TextView txtUnreadSMS = (TextView) v
                    .findViewById(R.id.txtUnreadSMS);
            txtUnreadSMS.setVisibility(View.GONE);
            // adapter 中 getItemId 返回的会话 ID
            Intent intent = new Intent(context, MessageListActivity.class);
            Bundle bundle = new Bundle();
            thread_id = (int) id;
            bundle.putInt("thread_id", thread_id);
            // bundle.putString("phone", broadcastPhone);
            intent.putExtras(bundle);
            startActivity(intent);
            messageManager.UpdateSMS((int) id);
        }
    });
    // 跳转到新建短信
    imgNewSMS.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent(context, NewMessageActivity.class);
            startActivity(intent);
        }
    });
```

```
        registerReceiver(smsReceiver, new IntentFilter(
                "android.provider.Telephony.SMS_RECEIVED"));
    }
    private SmsReceiver smsReceiver = new SmsReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            // Toast.makeText(context, "收到短信", Toast.LENGTH_SHORT).show();
            /* 以 Bundle 对象解开传来的参数 */
            Bundle bundle = intent.getExtras();
            /* 若 Bundle 对象不为空值,取出参数 */
            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");
                SmsMessage[] messages = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++) {
                    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
                for (SmsMessage message : messages) {
                    String strFrom = message.getDisplayOriginatingAddress();
                    String strMsg = message.getDisplayMessageBody();
                    // makeNotify(context, strFrom, strMsg);
                    refresh(strFrom, strMsg);
                    // Log.i("receive", "main 收到短信:" + strFrom);
                }
            }
            super.onReceive(context, intent);
        }
    };
    private void refresh(String strFrom, String strMsg) {
        if (isFromBroadcast)
            broadcastPhone = strFrom;
        int mThread_id = messageManager.getMessageThreadIdByPhone(strFrom);
        if (!broadcastPhone.equals(strFrom))
            isFromBroadcast = false;
        // Log.i("refresh", "thread_id: " + mThread_id);
        Message message = null;
        int position = 0;
        if (mThread_id == -1) {                         // 系统数据库中没有此 thread_id
            // Log.i("refresh", "系统数据库中没有此 thread_id ");
            for (Message sms : list) {
                if (strFrom.equals(sms.getPerson())) {    // UI 层的 list 中有此 thread_id
                    mThread_id = sms.getThread_id();
                    break;
                }
                position++;
            }
        } else {// 系统数据库中有此 thread_id
            // Log.i("refresh", "系统数据库中有此 thread_id");
            for (Message sms : list) {
                int thread_id = sms.getThread_id();
                if (mThread_id == thread_id) {
```

```
                    message = sms;
                    break;
                }
                position++;
            }
        }
        if (position > (list.size() - 1))
            position = 0;
        if (message != null) {// 系统中 thread_id 已存在,修改
            message.setBody(strMsg);
            message.setDate(messageManager.setDate(System.currentTimeMillis()
                    + ""));
            message.setRead(0);
            list.remove(position);
            list.add(0, message);
            position = 0;
        } else {// 系统中 thread_id 不存在,添加
            message = new Message();
            message.setPhone(strFrom);
            message.setBody(strMsg);
            message.setDate(messageManager.setDate(System.currentTimeMillis()
                    + ""));
            message.setRead(0);
            if (mThread_id == -1) {// UI 层的 list 中没有此 thread_id
                int thread_id;
                if (list.size() == 0) {
                    // Log.i("refresh", list + "");
                    thread_id = 1;
                } else {
                    thread_id = list.get(0).getThread_id() + 1;
                }
                message.setThread_id(thread_id);
                list.add(0, message);
            } else {// UI 层的 list 中有此 thread_id
                // Log.i("refresh", "UI 层的 list 中有此 thread_id");
                message.setThread_id(mThread_id);
                list.set(position, message);
            }
        }
        adapter.refreshFromBroadcast(list, true, position);
        // Log.i("receive", "message:" + message.toString());
    }
    public void bindListView() {
        list = messageManager.getMainSmsList();
        adapter = new MainListAdapter(context, list);
        listView.setAdapter(adapter);
    }
    private void getId() {
        imgNewSMS = (ImageView) findViewById(R.id.imgNewSMS);
        listView = (ListView) findViewById(R.id.mainList);
```

```
    }
    // 创建上下文菜单内容 STEP.2
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
            ContextMenuInfo menuInfo) {
        AdapterContextMenuInfo info = (AdapterContextMenuInfo) menuInfo;
        // adapter 中 getItemId 返回的会话 ID
        Message message = messageManager
                .getMainSmsListById((int) info.id, true).get(0);
        Contact contact = contactManager.getContactByNumber(message.getPhone());
        if (contact != null && !contact.getName().equals("")) {
            menu.setHeaderTitle(contact.getName());
        } else {
            menu.setHeaderTitle(message.getPhone());
        }
        if (contact == null)
            menu.add(0, 0, 0, "添加至通讯录");
        menu.add(0, 1, 1, "呼叫联系人");
        menu.add(0, 2, 2, "加入黑名单");
        menu.add(0, 3, 3, "删除该人的聊天对话");
        menu.add(0, 4, 4, "查看消息收藏夹");
        super.onCreateContextMenu(menu, v, menuInfo);
    }
    // menuItem 单击事件响应 STEP.3
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        AdapterContextMenuInfo info = (AdapterContextMenuInfo) item
                .getMenuInfo();
        // adapter 中 getItemId 返回的会话 ID
        Message message = messageManager
                .getMainSmsListById((int) info.id, true).get(0);
        Contact contact = contactManager.getContactByNumber(message.getPhone());
        String name;
        if (contact != null) {
            name = contact.getName();
            if (name.equals(""))
                name = message.getPhone();
        } else {
            name = message.getPhone();
        }
        switch (item.getItemId()) {
        case 0:// 添加至通讯录
                // ToastTool.showMessage(context, "添加至通讯录");
            break;
        case 1:// 呼叫联系人
                // ToastTool.showMessage(context, "呼叫联系人");
            break;
        case 2:// 加入黑名单
                // ToastTool.showMessage(context, "加入黑名单");
            break;
```

```
            case 3:// 删除该人的聊天对话
                createDelAlertDialog((int) info.id, name, info.position);
                // Log.i("refresh", info.position + "");
                break;
            case 4:// 查看消息收藏夹
            break;
            default:
                break;
            }
            return super.onContextItemSelected(item);
        }
        private void createDelAlertDialog(final int thread_id, String name,
                final int position) {
            new AlertDialog.Builder(context).setTitle("提示: ")
                    .setMessage("确定删除与" + name + "的全部对话?")
                    .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int which) {
                            messageManager.DeleteSMSByThreadId(thread_id);
                            list.remove(position);
                            // List<Message> list = messageManager.getMainSmsList();
                            adapter.refreshFromBroadcast(list, false, 0);
                        }
                    }).setNegativeButton("取消", null).create().show();
        }
        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
            if (keyCode == event.KEYCODE_BACK) {
                unregisterReceiver(smsReceiver);
            }
            return super.onKeyDown(keyCode, event);
        }
        @Override
        protected void onResume() {
            if (isFirst) {
                isFirst = false;
            } else {
                // 设置已读
                messageManager.UpdateSMS(thread_id);
                list = messageManager.getMainSmsList();
                Log.i("broadcast", "list:" + list.get(0).toString());
                adapter.refreshFromBroadcast(list, false, 0);
            }
            super.onResume();
        }
    }
```

参 考 文 献

[1] 田泽.嵌入式系统开发与应用.北京：北京航空航天出版社，2005.
[2] 杨光，孙丹.J2ME 程序设计实例教程.北京：清华大学出版社，2008.
[3] 张利国，刘伟.Java SE 应用程序设计.北京：科学出版社，2008.
[4] 余浩东.J2EE 应用框架设计与项目开发.北京：清华大学出版社，2008.
[5] Bruce Eckel.Java 编程思想.侯捷译.北京：机械工业出版社，2004.
[6] Shane Conder.Android 移动应用开发从入门到精通.北京：人民邮电出版社，2010.
[7] 高彩丽，徐黎明，袁海.Android 应用开发范例精解.北京：清华大学出版社，2012.
[8] Ed Brunette. Android 基础教程.北京：人民邮电出版社，2010.
[9] 蒋耘晨.Android 系统原理与实战应用.北京：北京理工大学出版社，2011.
[10] E2ECloud 工作室.深入浅出 Google Android.北京：人民邮电出版社，2011.
[11] 杨丰盛.Android 技术内幕：系统卷.北京：机械工业出版社，2011.
[12] 杨丰盛.Android 应用开发揭秘.北京：机械工业出版社，2010.
[13] 柯元旦，宋锐.Android 程序设计.北京：北京航空航天大学出版社，2010.
[14] 郭宏志.Android 应用开发详解.北京：电子工业出版社，2010.
[15] 王家林.Android 商业软件开发全程实战.北京：电子工业出版社，2011.
[16] 韩超.Android 系统原理及开发要点详解，北京：电子工业出版社，2010.
[17] 李宁.Android/Ophone 开发完全讲义.北京：中国水利水电出版社，2010.
[18] 陈平华，李文亮.Android 内核分析.广州：广东工业大学计算机学院，2010.
[19] 宋锐，柯元旦.Android 程序分析.北京：北京航空航天大学出版社，2010.
[20] 罗苑棠.嵌入式 Linux 驱动程序与系统开发实例精讲.北京：电子工业出版社，2009.
[21] 张光建，刘政.嵌入式 Linux 驱动程序开发实例教程.北京：清华大学出版社，2011.
[22] 王俊杰，曹丽.传感器技术与仪器.北京：清华大学出版社，2011.
[23] 沈玉龙，裴庆祺，马建峰.无线传感器网络安全技术概论.北京：人民邮电出版社，2010.
[24] 马祖长，孙怡宁，梅涛.无线传感器网络综述.通信学报，2004.
[25] 胡洪坡，宋孝先，张军.无线传感器网络综述.电信快报，2011.
[26] HEINZELMANW，CHANDRAKASANA，BALAKRISHNAN H. Energy-efficient communication protocol for wireless microsensor networks. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences. Hawaii，USA，2000.
[27] SOHRABI K，POTTIE G. Performance of a novel self-organization protocol for wireless ad hoc sensor networks. IEEE 50th Vehicular Technology Conference. Amsterdam，Netherlands，1999.
[28] GONG Lei，ZHOU Cong. Development and research of Mobile Termination Application Based on Android. Computer and Modernization，2008，8(1).
[29] J. P. Bastard，L. PiCroni. Plasma plasminogen activator inhibitor 1，insulin resistance and android obesity.
[30] Ming-Chao Chen，Jian-Liang Chen，Teng-Wen Chang. Android/OSGi-based vehicular network management system.
[31] Woo Park，Jung-Yup Kim，Baek-Kyu Cho，Jun-Ho Oh. Control hardware integration of a biped humanoid robot with an android head.
[32] 邓凡平.深入理解 Android(卷 1).北京：机械工业出版社，2006.

[33] 梅尔(Reto Meier),王超. Android 2 高级编程. 2 版. 北京. 清华大学出版社,2008.
[34] 李佐彬,等. Android 开发入门与实战体验. 北京：机械工业出版社,2008.
[35] Alessandro Distefano, Gianluigi Me, Francesco Pace. Android anti-forensics through a local paradigm. America：ACM Programming,2006.
[36] Sean J. Barbeau, Miguel. Labrador, Philip L. Winters, Rafael Pe'rez, Nevine Labib Georggi. Location API 2.0 for J2ME-A new standard in location for Java-enabled mobile phones. America, Germany. ACM Programming,2008.
[37] 韩超,梁泉. Android 系统原理及开发要点详解. 北京：电子工业出版社,2010.